HP Virtual User Generator

for the Windows operating system

Software Version: 9.50

User Guide

Document Release Date: January 2009
Software Release Date: January 2009

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

Trademark Notices

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

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 guide'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://h20230.www2.hp.com/selfsolve/manuals

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

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

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

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

Support

You can visit the HP Software Support web site at:

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

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

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

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

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

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

To register for an HP Passport ID, go to:

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

Table of Contents

Welcome to This Guide

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

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

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

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

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

This chapter includes:

How This Guide Is Organized

This guide contains the following parts:

Part I Working with VuGen

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

Part II Protocols

Provides information relevant to individual protocols and groups of protocols.

Part III Parameters

Describes the process of creating parameters. Provides information about parameter types and properties.

Part IV Recording Options

Provides information about the recording options relevant to all protocols as well as selected individual protocols.

Part V Run-Time Settings

Provides information about the run-time settings relevant to all protocols as well as selected individual protocols.

Part VI Information for Advanced Users

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

Part VII 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.

Who Should Read This Guide

This guide is for the following users:

- Script developers
- Functional Testers
- Load Testers

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

LoadRunner Online Documentation

LoadRunner includes a complete set of documentation describing how to use the product. The documentation is available from the help menu and in PDF format. PDFs can be read and printed using Adobe Reader, which can be downloaded from the Adobe Web site *(*http://www.adobe.com*)*. Printed documentation is also available on demand.

Accessing the Documentation

You can access the documentation as follows:

- From the **Start** menu, click **Start** > **LoadRunner** > **Documentation** and select the relevant document.
- From the **Help** menu, click **Documentation Library** to open the merged help.

Getting Started Documentation

- **Readme.** Provides last-minute news and information about LoadRunner. You access the Readme from the **Start** menu.
- **HP LoadRunner Quick Start** provides a short, step-by-step overview and introduction to using LoadRunner. To access the Quick Start from the Start menu, click **Start** > **LoadRunner** > **Quick Start.**

- **HP LoadRunner Tutorial.** Self-paced printable guide, designed to lead you through the process of load testing and familiarize you with the LoadRunner testing environment. To access the tutorial from the Start menu, click **Start** > **LoadRunner** > **Tutorial.**

LoadRunner Guides

- **HP Virtual User Generator User Guide.** Describes how to create scripts using VuGen. The printed version consists of two volumes, Volume I - *Using VuGen* and Volume II - *Protocols*, while the online version is a single volume. When necessary, supplement this user guide with the online *HP LoadRunner Online Function Reference*.
- **HP LoadRunner Controller User Guide.** Describes how to create and run LoadRunner scenarios using the LoadRunner Controller in a Windows environment.
- **HP LoadRunner Monitor Reference.** Describes how to set up the server monitor environment and configure LoadRunner monitors for monitoring data generated during a scenario.
- **HP LoadRunner Analysis User Guide.** Describes how to use the LoadRunner Analysis graphs and reports after running a scenario to analyze system performance.
- **HP LoadRunner Installation Guide.** Explains how to install LoadRunner and additional LoadRunner components, including LoadRunner samples.

LoadRunner References

- **LoadRunner 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.
- **Analysis API Reference.** This Analysis API set can be used for unattended creating of an Analysis session or for custom extraction of data from the results of a test run under the Controller. You can access this reference from the Analysis Help menu.

- **LoadRunner Controller Automation COM and Monitor Automation Reference.** An interface with which you can write programs to run the LoadRunner Controller and perform most of the actions available in the Controller user interface. You access this reference (**automation.chm)** from the <**LoadRunner Installation**>/**bin** directory.
- **Error Codes and Troubleshooting.** Provides clear explanations and troubleshooting tips for Controller connectivity and Web protocol errors. It also provides general troubleshooting tips for Winsock, SAPGUI, and Citrix protocols.

Additional Online Resources

For additional help, refer to one of the following resources:

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

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help** > **HP Software Support**. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:
http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport user ID, go to:
http://h20229.www2.hp.com/passport-registration.html

HP Software Web site accesses the HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. Choose **Help** > **HP Software Web site**. The URL for this Web site is www.hp.com/go/software.

Documentation Updates

HP Software is continually updating its product documentation with new information.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (http://h20230.www2.hp.com/selfsolve/manuals).

Part 1

Working with VuGen

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 the Virtual User Generator, *VuGen*.

This chapter includes:

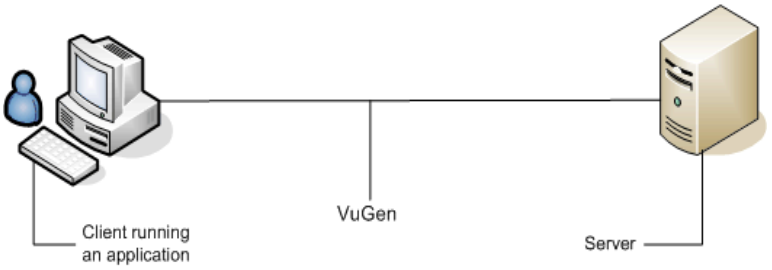
Note: The online version of the *Virtual User Generator* guide is a single volume, while the printed version consists of two volumes, *Volume I-Using VuGen* and *Volume II-Protocols*.

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.

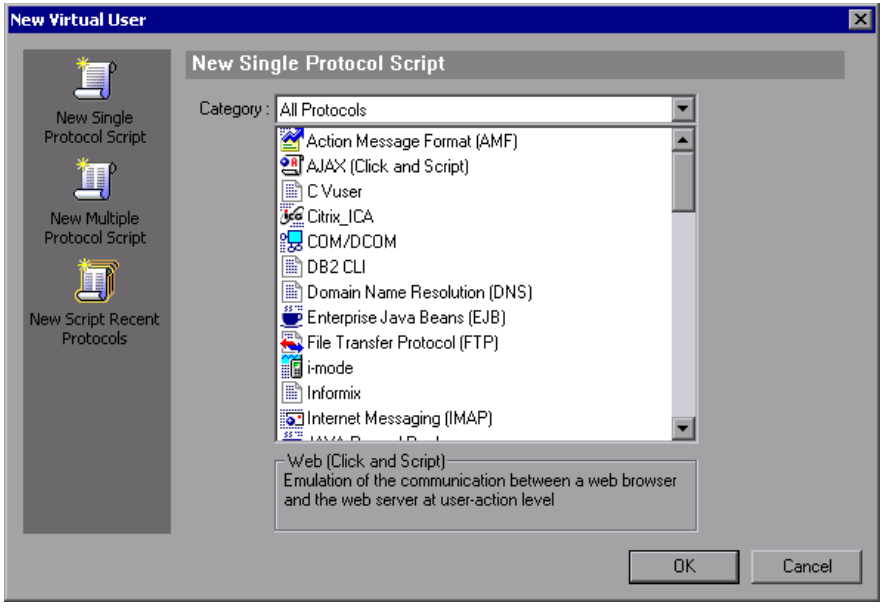


During playback, Vuser scripts communicate directly with the server by executing calls to the server API. When a Vuser communicates directly with a server, system resources are not required for the client interface. This lets you run a large number of Vusers simultaneously on a single workstation, and 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 template, Java template, Javascript, VB script, VBNet, and VB template type scripts
- **Distributed Components.** For COM/DCOM, and Microsoft .NET protocols
- **E-business.** For AJAX (Click and Script), AMF, Flex, FTP, LDAP, Microsoft .NET, Web (Click and Script), Web (HTTP/HTML), and Web Services protocols
- **ERP/CRM.** For Oracle NCA, Oracle Web Applications 11i, Peoplesoft Enterprise, Peoplesoft-Tuxedo, SAP-Web, SAPGUI, SAP (Click and Script), and Siebel Web protocols
- **Enterprise Java Beans.** For EJB Testing
- **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 protocol
- **Streaming.** For MediaPlayer (MMS) and RealPlayer protocols

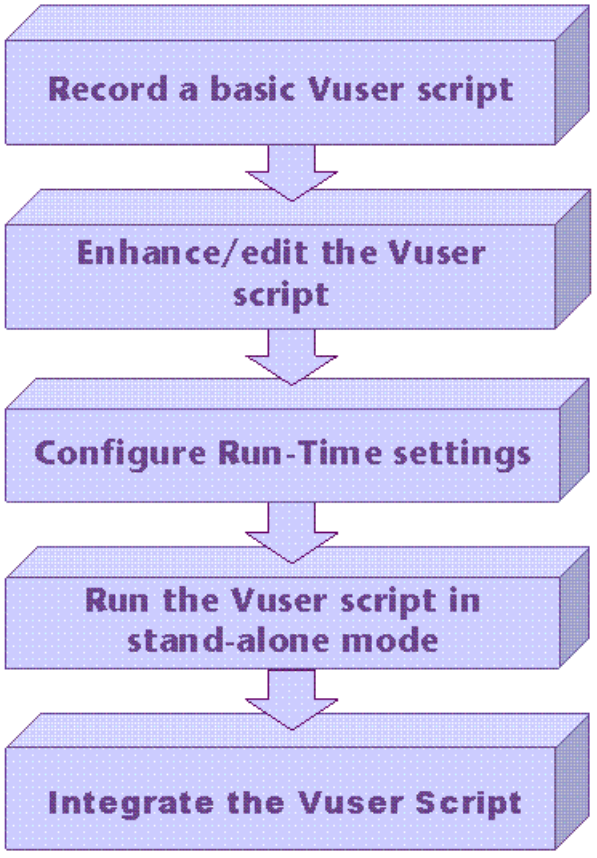
➤ **Wireless.** For Multimedia Messaging Service (MM) and WAP protocols

To view a list of all supported protocols in alphabetical order, select **File** > **New** and select *All Protocols* in the **Category** 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, select **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, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

HP LoadRunner Licensing

When you purchase LoadRunner, you receive a custom license for your configuration and needs. You can access this information at any time through the LoadRunner Launcher. To preview your license key information, choose **Start** > **Programs** > **LoadRunner** to open the LoadRunner Launcher. Select **LoadRunner License** from the Launcher's **Configuration** menu.

When you purchase a license for Service Test, you install it through the License Manager (**Start** > **Programs** > **LoadRunner** > **Service Test** > **License Manager**). The license can be a Seat license (host lock) or a Concurrent license (site license). For more information, see the *HP LoadRunner Installation Guide*.

HP Service Test is available as a standalone product or as an extension to HP LoadRunner's Virtual User Generator, VuGen. Service Test contains all of the capabilities of VuGen with added functionality in the area of SOA and Web Service testing.

HP Service Test is a functional testing tool and therefore provides more functional testing features than VuGen. In addition, Service Test contains several utilities that allow working with WSDLs, XML, and SOAP.

All scripts created in HP Service Test can run in HP LoadRunner and HP Performance Center. This compatibility exists only when Service Test and LoadRunner are of the same version.

LoadRunner and Service Test

The following features exist only in HP Service Test (standalone) or LoadRunner with Service Test:

Functional Testing Utilities

- **XML Validation.** checks if XML is well-formed, complies with a schema, and uses expected values.
- **Checkpoints.** allows you to compare the response with expected values.
- **Negative Testing.** the ability to define SOAP faults as the desired response, enabling you to test error cases.
- **WS-I Validation.** the ability to check if the WSDL and SOAP are WS-I compliant
- **Test Generation Wizard.** a wizard guiding you in creating aspect-based tests

Integration with Quality Center

- **BPT.** Business Process Testing. Service Test can create components for Quality Center's BPT and run them in a business scenario.
- **Remote execution from QC.** Service Test scripts can be launched directly from Quality Center. You can define the parameters and runtime settings, run the test, and inspect the results from within Quality Center.

Other Tools

- **Service Emulation.** a tool to create an emulated service to simulate both a client and server
- **Command line invocation.** the ability to invoke a script from the command line

2

Introducing VuGen

The Virtual User Generator, also known as VuGen, helps you develop Vuser scripts for a variety of application types and communication protocols.

This chapter includes:

About VuGen

The Virtual User Generator, also known as 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.

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.

VuGen records Vuser scripts on Windows platforms only. However, a recorded Vuser script can be run on both Windows and UNIX platforms.

Starting VuGen

VuGen can be run from a number of different applications such as HP Business Availability Center, HP LoadRunner, and HP Performance Center.

To start VuGen, select **Start** > **Programs** > <*App_Name*> > **Applications** > **Virtual User Generator**. The Virtual User Generator Start Page opens.

The following actions are available from the start page:

Action	Procedure
To open an existing script	Click **Open Existing Script**
To open an existing script from a zip archive	Select **File** > **Zip Operations** > **Import From Zip File**
To create a new script	Click **New Vuser Script**

Customizing the Commands

You can customize the appearance of VuGen by configuring toolbars and shortcuts from the Customize dialog box using one of the following tabs:

- Commands Tab
- Toolbars Tab
- Tools Tab
- Keyboard Tab
- Options Tab

Commands Tab

Select a toolbar in the **Categories** section (left pane). The toolbar's buttons are displayed in the right pane, **Commands**. Click on a command to view its description. Drag the desired items from the right pane into the desired open toolbar.

Toolbars Tab

The **Toolbars** tab lets you indicate which toolbars to display. Select the check box for each toolbar you want to display. The available toolbars are Standard, Edit, View, Record, Vuser, Debug, Tools, Tree, and Advanced.

Additionally, the following buttons in this tab allow you to configure the toolbar settings:

- **Reset.** Returns to the default toolbar view settings for toolbar selected in the left pane.
- **Reset All.** Returns to the default toolbar view settings for all toolbars. VuGen issues a warning before reverting back to the default settings.
- **New.** Adds a custom toolbar to the list of shown toolbars. Drag this toolbar to the desired location in the VuGen window. Afterwards, use the Commands tab to add icons to the toolbar.
- **Rename.** Allows you to rename the selected toolbar. This button is only enabled when you select a custom toolbar.
- **Delete.** Deletes the selected toolbar. This button is only enabled when you select a custom toolbar.
- **Show text labels.** Shows text labels for the selected toolbar. For example, if you enable this option for the Record toolbar, the Stop button will display the word Stop and the Run button will display Run.

Tools Tab

The **Tools** tab lets you add custom commands to VuGen toolbars. The command is usually an executable, batch, or *.com* file. In addition you can indicate a specific document or PDF file that you may want to open from within VuGen. You define one or more tools in the Tools tab Tool list. VuGen lists these tools under the Commands tab **Tools** menu. You select the desired command from the Tool menu, and drag it onto the desired toolbar. For example, you can specify *calc.exe*, the Windows calculator, to allow you to open it from VuGen.

Menu Contents

➤ **New.** (left button) Add an item to the Menu Contents list. VuGen creates a new item in the Menu contents list and the cursor moves to the beginning of the line. Type in the name of the command.

➤ **Delete.** (second button) Deletes the selected item from the Menu Contents list.

Up Arrow / Down Arrow. Move the selected item up or down.

Item Details

➤ **Command.** Type in the name of an executable or batch file. Alternatively, click the Browse button to the right of the field, and locate the desired file. If you specify a non-executable file, make sure that the file extension is associated with a program that is installed on the machine.

➤ **Arguments.** Specify run-time arguments to be executed with the specified file.

➤ **Initial Directory.** The initial directory in which to run the custom tool.

Keyboard Tab

The Keyboard tab lets you assign keyboard shortcuts to menu commands. You can assign shortcuts to both standard and custom commands.

To assign a keyboard shortcut:

1 Select a menu category from the Category box.

2 Select a command within the chosen category from the **Commands** box.

3 Click in the **Press New Shortcut Key** box and press the shortcut key or key combination. VuGen displays the key or key combination in the box.

4 Click **Assign**.

5 To remove an assignment, select the command and view its assigned shortcuts. Select the shortcut you want to remove in the **Current Keys** box, and click **Remove**.

6 To restore all default assignments, click **Reset All**. Note that this also deletes all custom keyboard assignments.

Options Tab

Toolbar

You can set several display options that apply to all of the toolbars:

- **Show ScreenTips on toolbars:** Show the screen tips when moving the cursor over the toolbar buttons (enabled by default).
  - **Show Shortcut Keys in Screen Tips:** Show the keyboard shortcut in the tooltips (disabled by default).
- **Large Icons:** Display large buttons on the toolbar (disabled by default).

Personalized Menu and Toolbars

You can personalize the way commands are shown in the menus and toolbars:

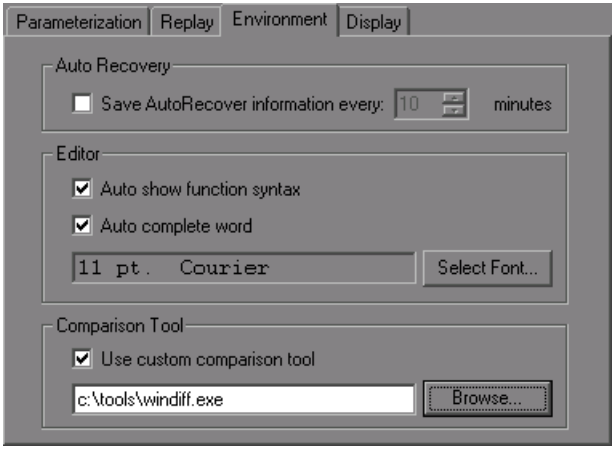
- **Menus show recently used commands first:** The menus show the commands used most recently, at the top of the menu (enabled by default).
  - **Show full menu after a short delay:** For expandable menus, show the full menu after a short delay.

Restore Defaults

➤ **Reset my usage data.** Delete all of the custom commands and restore the default set of visible commands to the menus and toolbars. Does not undo any explicit customization.

Understanding the VuGen Environment Options

You can set up your VuGen working environment in order to customize the auto recovery and editor settings. You set these options from the Tools > 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 this function globally, select the check box. If you do not check this box, you can still enable this function manually 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 select the exact function without having to manually type in the entire function. To enable this function globally, select the check box. If you do not check this box, you can enable this function manually 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, etc.) are available.

Default Environment Settings

Show Function Syntax and **Auto complete word** are enabled globally by default. Auto Recovery is set to 10 seconds. If you manually changed these settings, you can restore the default values by clicking **Use Defaults**.

Comparison Tool

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

To compare Vuser scripts:

1. Open the first Vuser script that you want to compare.
2. Select **Tools** > **Compare with Script.**
3. Select the second Vuser script. The Vuser scripts are displayed in a new window, side by side. Differences are highlighted in yellow.

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. 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.
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 edit 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 66.

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

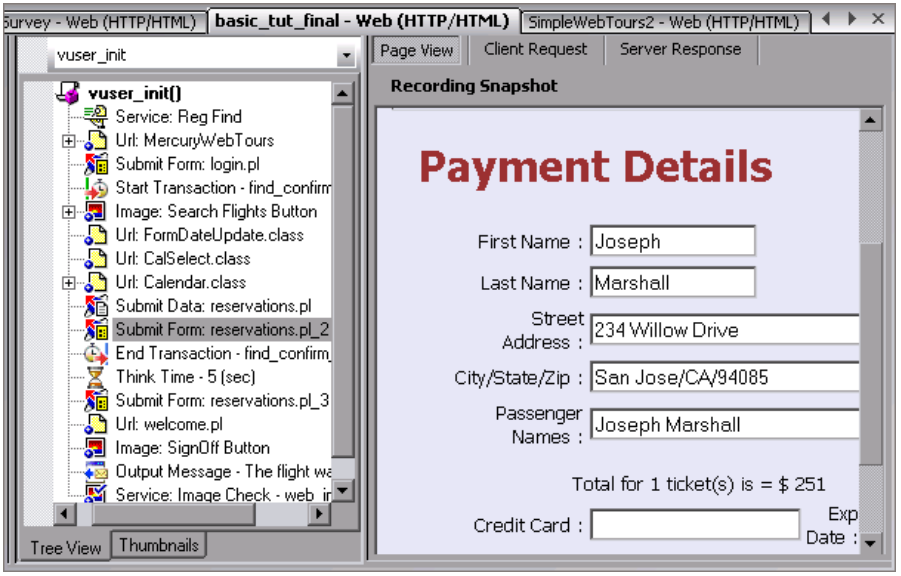
VuGen's Tree view shows the Vuser script in an icon-based format, with each step represented by a different icon.

To display the Tree view:

From VuGen's main menu, select **View** > **Tree View**, or click the **View Tree** button.

The Actions section of the Vuser script is displayed in the icon-based Tree view. To display a different section, select 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 Select **Insert Before** or **Insert After** from the right-click menu. The Add Step dialog box opens.

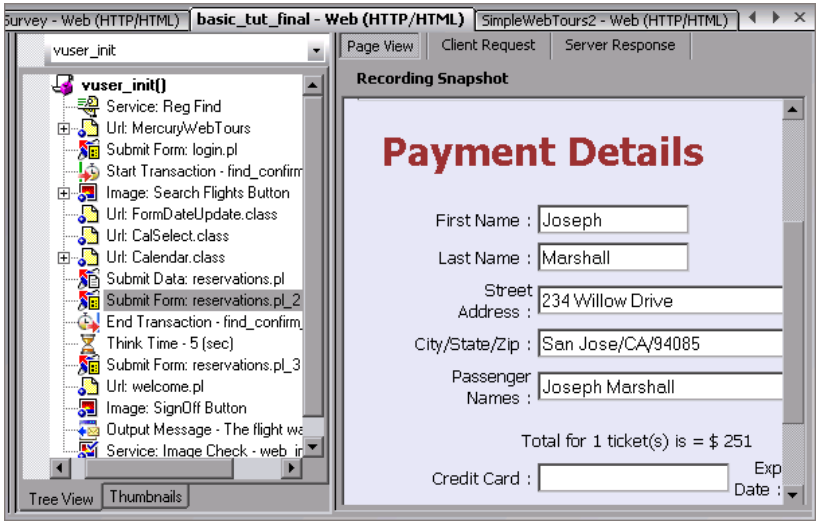
3 Select a step and click **OK**. The Properties dialog box opens.

4 Specify the properties and click **OK**. VuGen inserts the step before or after the current step.

Note: Inserting a step after a parent node results in the detachment of the children nodes from the parent node.

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 the section on correlation after recordng.

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. Select **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, select a different snapshot for comparison. To select a specific replay snapshot, select the expanded menu of **View > Snapshot > Select Iteration**. Select a set of results and click **OK.**

Multiple Snapshots

In several protocols such as Microsoft Remote Desktop Protocol (RDP), you can view multiple snapshots for a single step. This occurs when a mismatch occurs during replay and you choose to append the new image to the step. For more information, see the RDP protocol section.

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 .png files in the script's directory. To determine the name of the snapshot file, check the function's arguments in Script view.

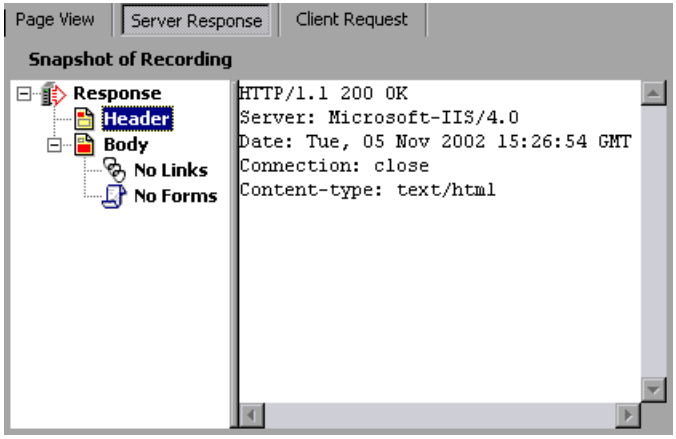
```
ctrx_sync_on_window("ICA Administrator Toolbar", ACTIVATE, 768, 0, 33,
                573, "snapshot12", CTRX_LAST);
```

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 select **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

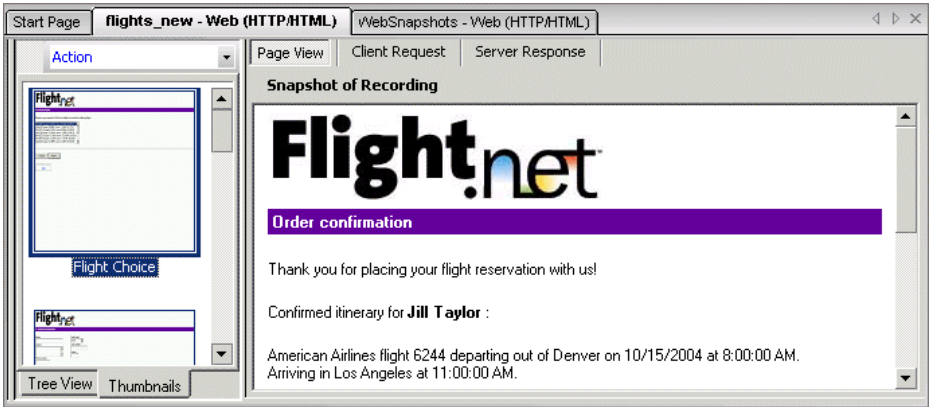
In Tree view, the **Thumbnail** tab appears at the bottom of the Tree view window.

By default, the thumbnail view only shows primary steps in your script. To show all thumbnails, select **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, select **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

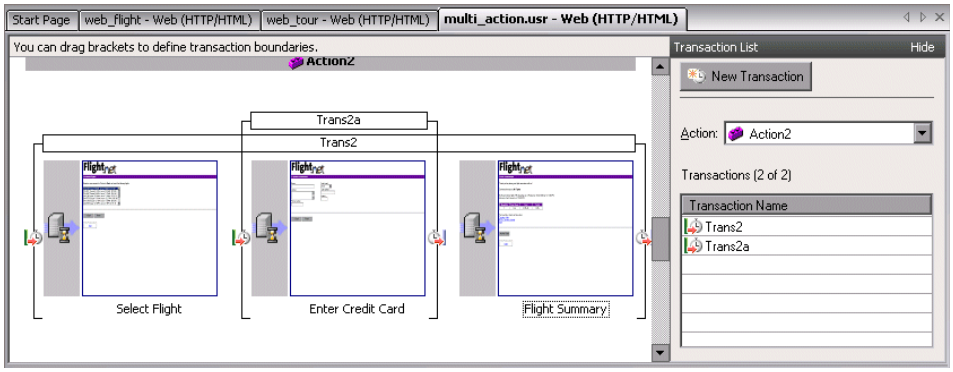
You can view the snapshots through the Transaction Editor. This view sorts the thumbnails by actions and provides you with an flat thumbnail view of all of the script's steps.

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:

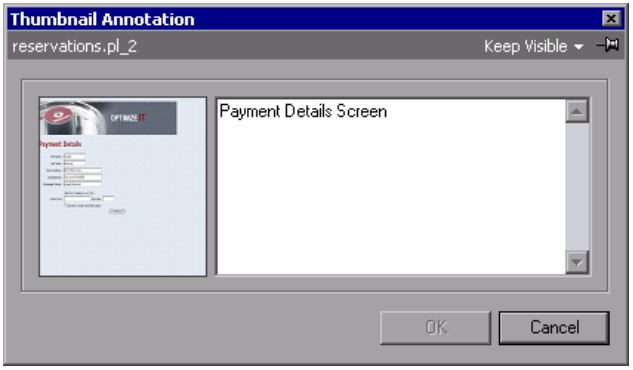
Select **View Larger Image** from the right-click menu or press **Alt+F6**. A separate window opens showing a larger view of the thumbnail.

To rename a thumbnail:

1. Select the thumbnail and select **Rename** from the right-click menu or press F2.
2. Type in the desired text.

To annotate a thumbnail:

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

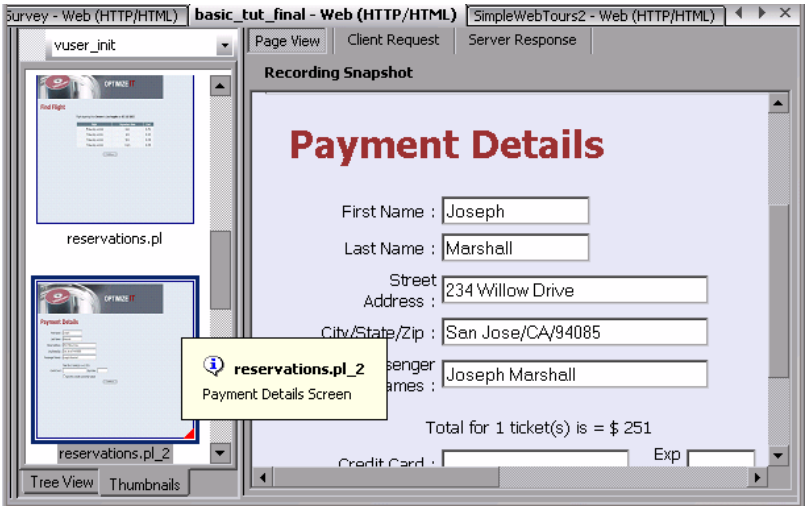


2 Type text into the right pane of the Thumbnail Annotation dialog box.

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

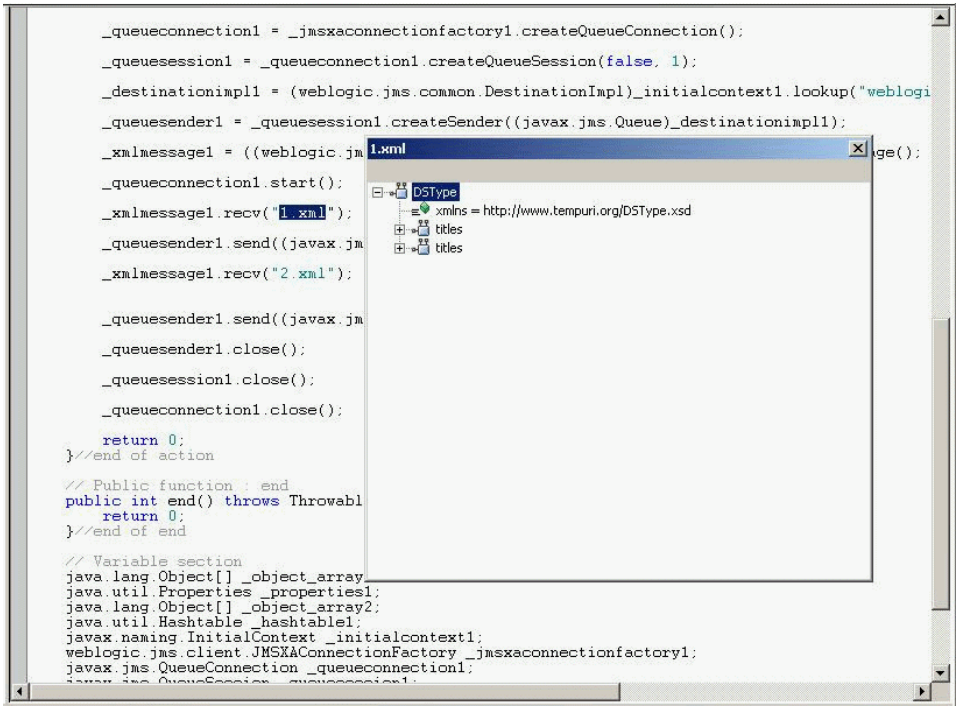
To leave the Annotation box open in order to add more text or work with other snapshots, select **Keep Visible** from the upper right corner. The **OK** button changes to **Apply**.

After you insert an annotation for a thumbnail, VuGen places a red mark in the bottom right corner 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 select **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 9, "Running Vuser Scripts in Standalone 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, see 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 79, "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 86, "Calling External Functions."

Advanced users can compile a recorded script to create an executable program. For more information, see Chapter 6, "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, see 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 6, "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 66 for a list of LR functions, and for details see 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 in Java for Java type protocols. You can instruct VuGen to generate code in Visual Basic or Javascript. For more information, see Chapter 76, "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 6, "Enhancing Vuser Scripts." For more information about modifying a Java script, see Chapter 38, "Programming Java Scripts."

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 6, "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 163 to search all of the files in the script.

Online Function Reference

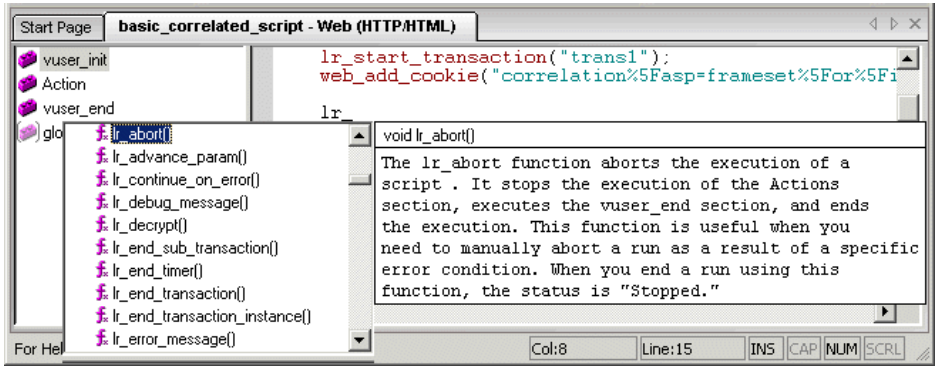
The *Online Function Reference* contains detailed syntax information about all of the VuGen functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

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

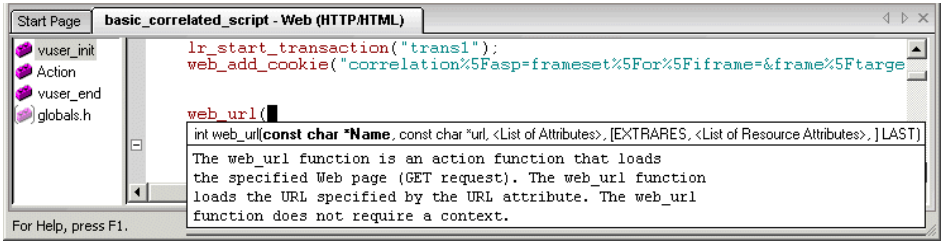


To use one of the displayed functions, select it, or scroll to the desired item and then select it. VuGen inserts the function at the location of the cursor. To close the list box, press the Esc key.

By default, VuGen uses word completion globally. To disable word completion, select **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, select **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 and Script)	**web_ajax.h**
Citrix	**ctrxfuncs.h**
COM/DCOM	**lrc.h**
Database	**lrd.h**
FTP	**mic_ftp.h**
General C function	**lrun.h**
IMAP	**mic_imap.h**
LDAP	**mic_mldap.h**
MAPI	**mic_mapi.h**
Oracle NCA	**orafuncs.h**
POP3	**mic_pop3.h**
RDP	**lrrdp.h**
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

Using the Protocol Advisor

VuGen provides a variety of protocols that can be used to create Vuser scripts. The Protocol Advisor analyzes your business processes and provides guidance for selecting a protocol to record your script.

This chapter includes:

About the Protocol Advisor

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

In most cases, more than one protocol is suggested and displayed, along with combinations of protocols. The following section contains guidelines on how to use the list of suggested protocols.

Protocol Advisor Workflow

The following steps represent a typical workflow for finding the optimal protocol to record your application using the Protocol Advisor:

1 Run the Protocol Advisor

a From the start page, select **File** > **Protocol Advisor** > **Analyze Application** or click the **Protocol Advisor** button. The Protocol Advisor dialog box opens.

b Fill in the fields as follows:

- **Application type.** Select the type of application.
- **Program to analyze.** Select the program to analyze.
- **URL Address.** This field is for Internet Applications only. Specify the starting URL address.
- **Program Arguments.** This field is for Win32 applications only. Specify command line arguments for the executable specified above. For example, if you specify **plus32.exe** with the program arguments **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. The required information differs, depending on the type of Vuser script.

c Click **OK**. A box is displayed showing the progress of the analysis.

d Perform typical business processes in your application. Try to walk through a variety of business processes to make sure that your results are comprehensive.

e Click **Stop Analyzing** to end the analysis.

The Protocol Advisor Results page displays the results.

2 Save the results.

Select **File** > **Protocol Advisor** > **Save Results**. Enter a name and select the directory.

3 Select the protocol to use.

We recommend selecting the protocols using the following order of priority:

- Multi/Combination protocol
- The highest level application protocol (see the diagram in step 6 on page 74)
- The first protocol on the list

4 Create a new Vuser script.

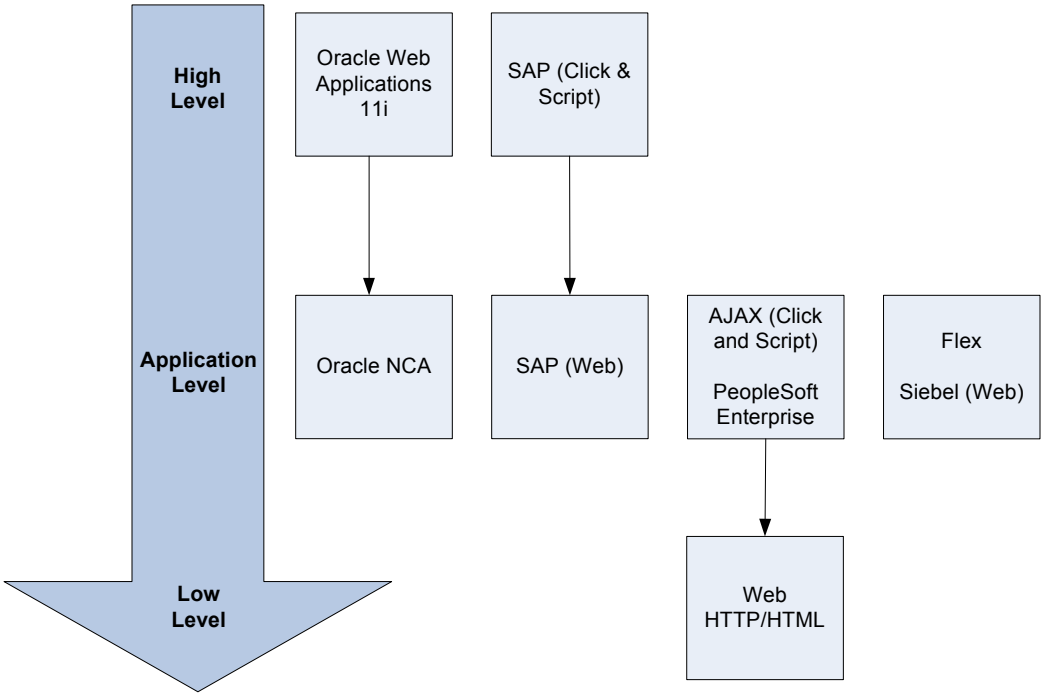
Create a new Vuser script according to the sections about the selected protocol.

5 Enhance, debug, and verify replay.

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

6 If you cannot replay your script, select a different protocol.

- **For all non Web-based protocols:** Return to the Protocol Advisor Results page and select the next protocol on the list and continue from step 2. If there are no more protocols listed, contact HP Customer Support.
- **For Web-based protocols:** The following diagram illustrates the Web-based protocols arranged according to their application level. The arrows indicate the order in which you should attempt a new protocol. For example, if the first protocol you used was Oracle Web Applications 11i and it failed to successfully replay, you would then try the Oracle NCA protocol.



Notes and Limitations

The following notes and limitations should be considered:

- This feature will only detect protocols supported by LoadRunner.
- Some Web protocols are detected based on the URL. For example, the URL may contain keywords such as SAP. Therefore, if you use a different URL or a different application with the same URL, the results may differ.
- The following protocols are frequently detected but are not always appropriate for use. You should only use them if there are no other detected protocols.
  - COM/DCOM
  - Java
  - .Net
  - WinSocket
  - LDAP

4

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.

This chapter includes:

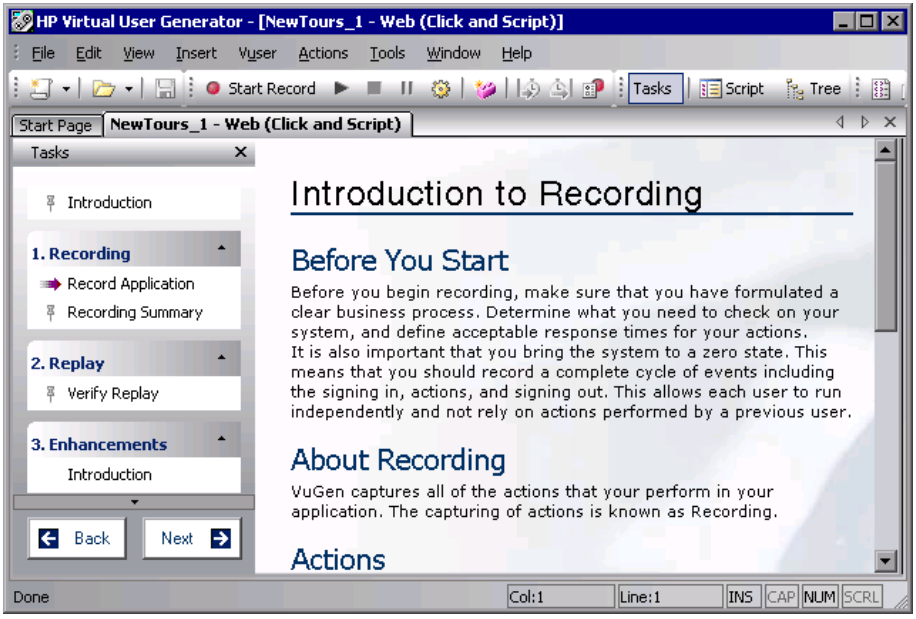
About Viewing the VuGen Workflow

VuGen's workflow screens walks you through the different steps of creating a script.

You can also work in Script View or Tree 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), **Replay**, **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))
Replay	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: Record Application and Recording Summary.

Record Application

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

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

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

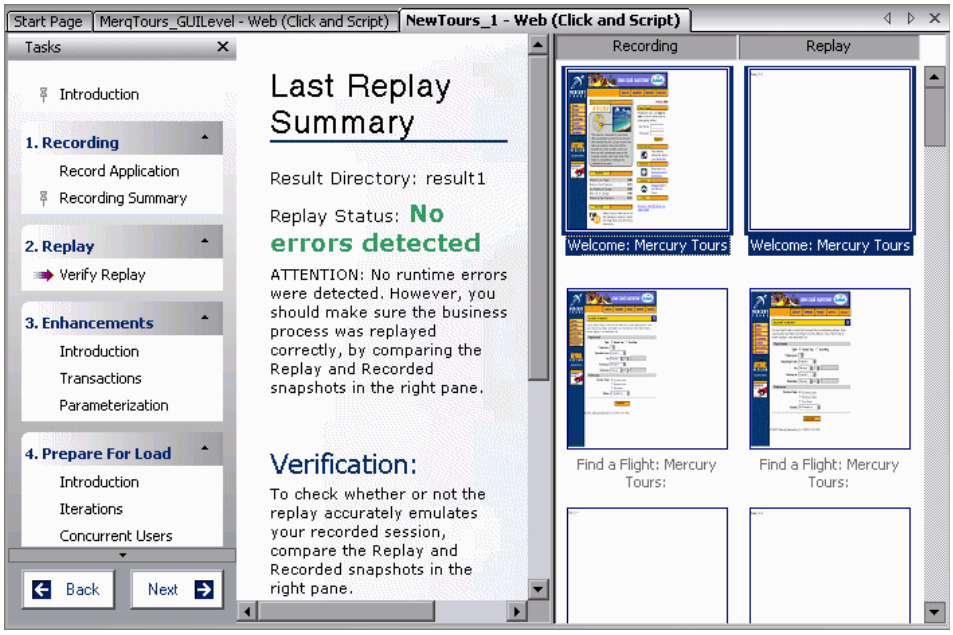
This wizard step provides an introduction to verification and contains the following sections.

- 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, select **View** > **Tree View** to switch to Tree view. Then select **View** > **Snapshot** > **Select Iteration** and select the desired iteration.

Enhancing the Script

The Enhancement Section has three primary steps:

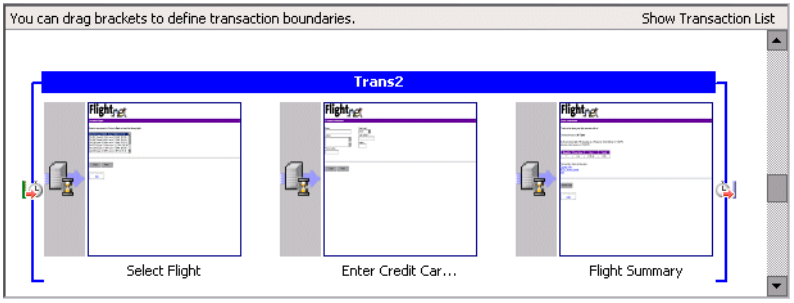
- Transactions
- Parameterization
- Content Check.

Transactions

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

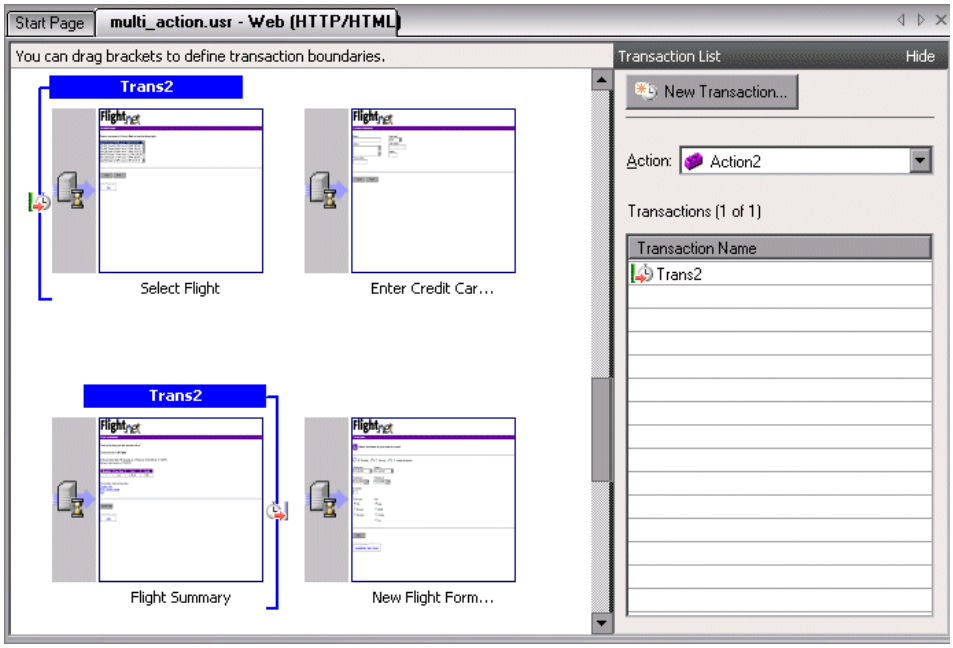
By default, VuGen only displays thumbnails for the primary steps in your script. To display thumbnails for all of the steps in your script, select **View** > **Show All Thumbnails**.

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

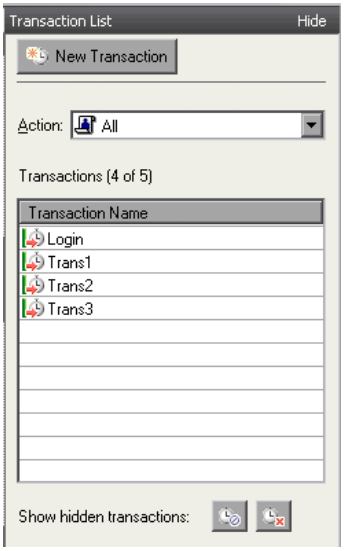


Working with the Transactions List

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 a caption similar to 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. Select **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 starting and ending thumbnails of the transaction.

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 select **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 select **New Transaction** from the right-click menu or click the **New Transaction** button in the top of the right pane.
5. VuGen displays instructions in the status area above the thumbnails as follows:

   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.

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

➤ To delete a transaction, select its title in the right pane and select **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.

➤ 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 for specific text or content in replayed web pages.

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.

Restoring the Workflow View

If your VuGen view no longer display the Workflow views, you can easily return to it:

➤ Make sure the **Tasks** Pane is visible in the left pane. If not, click the **Tasks** button on the toolbar.

➤ Click the top link, **Introduction**. The right pane displays the first Workflow screen.

Tip: To instruct VuGen to display the Workflow view after every replay, open the **Replay** tab (**Tools** > **General Options**) and select the **Replay summary** option in the After Replay section.

5

Recording with VuGen

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

This chapter includes:

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 6, "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 79, "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 Chapter 38, "Programming Java Scripts."

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

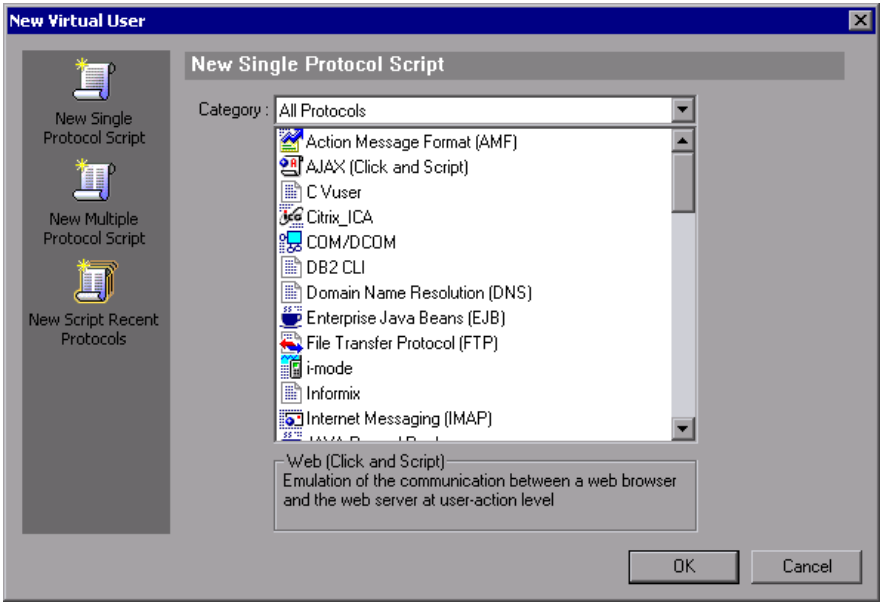
Note: Transaction Breakdown for Oracle DB is not available for actions recorded in the vuser_init section.

Creating New Virtual User Scripts

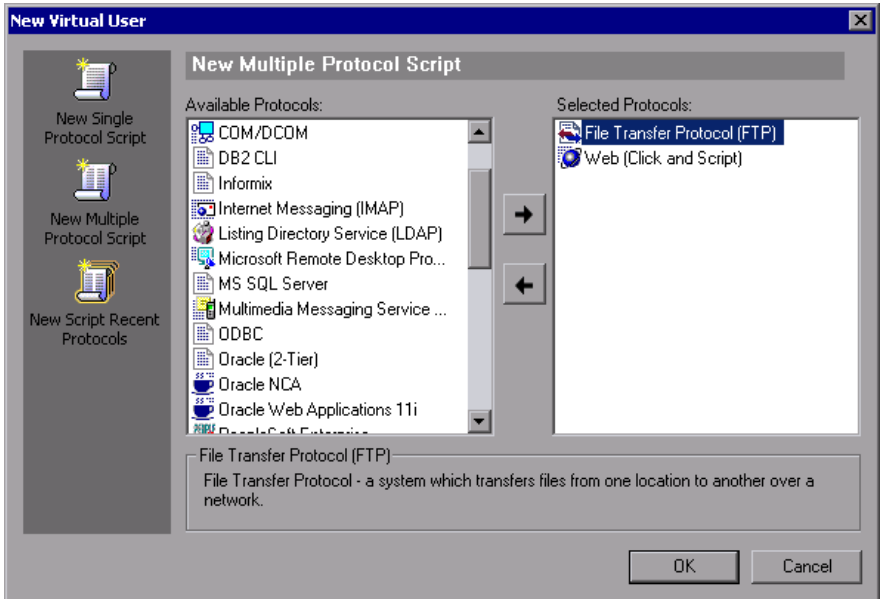
VuGen allows you to create new scripts by recording in either single or multi-protocol mode. It also provides a solution for creating scripts in a Service Oriented Architecture (SOA) environment.

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

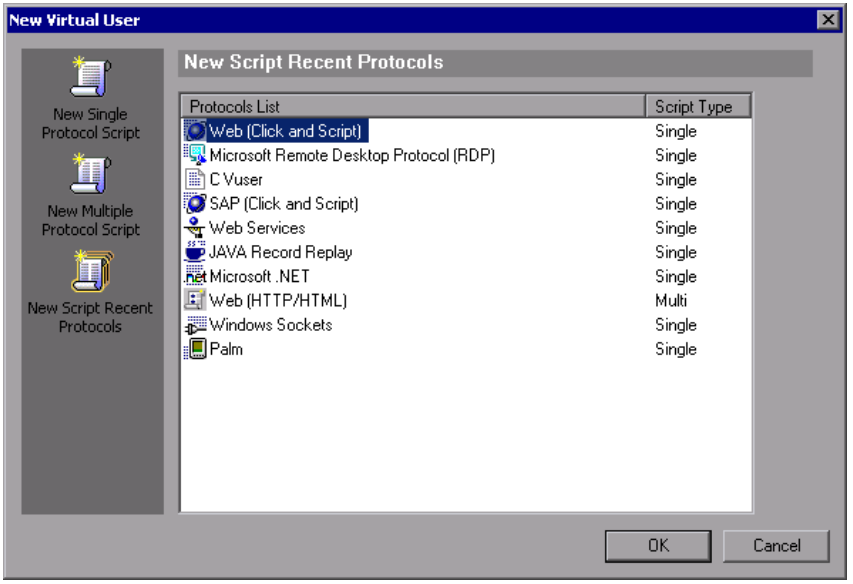
New Single Protocol Script. Creates a single protocol Vuser script. Select a category from the Category list (see "Choosing a Virtual User Category" on page 98), and select a protocol in the protocol list under that category.



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, select 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, Web, WAP, i-mode, 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 Chapter 25, "Java Protocols - Recording".

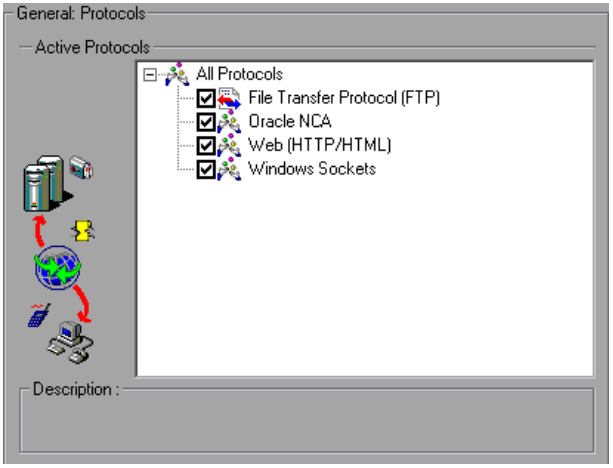
In SOA (Service Oriented Architecture) systems, it is essential that you test the stability of your applications and services before deployment. **LoadRunner VuGen** allows you to create basic Web Service scripts. **HP LoadRunner with Service Test**, HP's SOA testing tool, contains additional features that help you create a comprehensive testing solution for your SOA environment. For more information about **Service Test**, contact an HP representative.

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, select **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 "Setting the AMF Recording Mode" on page 1142.

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 Template, Visual Basic template, Java template, VBNet, Javascript, and VBscript type scripts
- **Distributed Components.** For COM/DCOM and Microsoft .NET protocols

- **E-Business.** For AMF, AJAX (Click and Script), Flex, FTP, LDAP, Microsoft .NET, Web (Click and Script), Web (HTTP/HTML), and the Web Services/SOA protocols
- **ERP/CRM.** For Oracle NCA, Oracle Web Applications 11i, Peoplesoft Enterprise, Peoplesoft-Tuxedo, SAP-Web, SAPGUI, SAP (Click and Script), and Siebel Web protocols
- **Enterprise Java Beans.** For the EJB Testing protocol
- **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 the Tuxedo protocol
- **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.
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 select **Oracle NCA**—not **Web Protocol**. For details, see Chapter 56, "Oracle NCA Protocol."
4. 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 99.

Opening an Existing Script

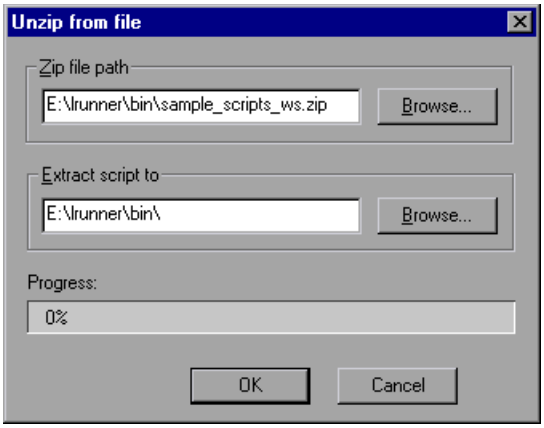
If you already have a script on the local machine or network, you can modify it and record additional actions.

To open an existing script:

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

2 To open a file from a Quality Center repository (LoadRunner only), see "Opening Scripts from a Quality Center Project" on page 204.

3 To open a script stored in a compressed zip file, select **File > Zip Operations > Import from Zip File**. After you select 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, select **File > Zip Operations > Work from Zip File**. When you modify the script and save it, the changes are stored directly in the zip file.

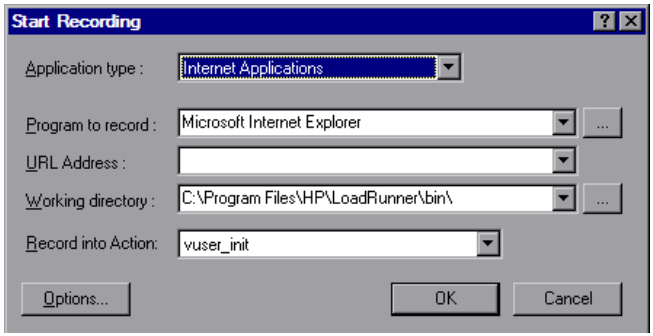
Recording Your Application

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.

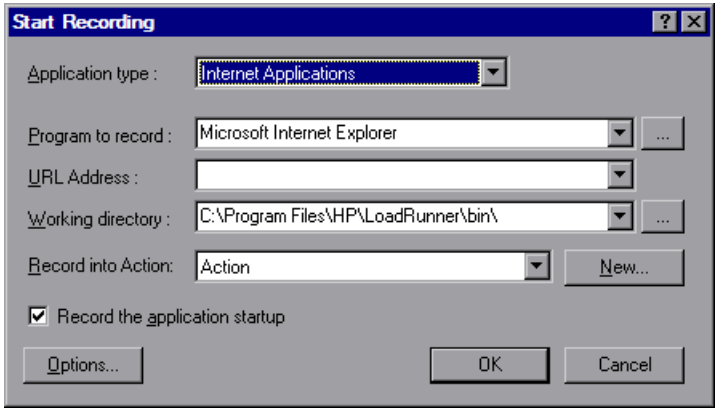
2 For most Client/Server protocols, the following dialog box opens:



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, select 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-based applications, fill in the relevant information:

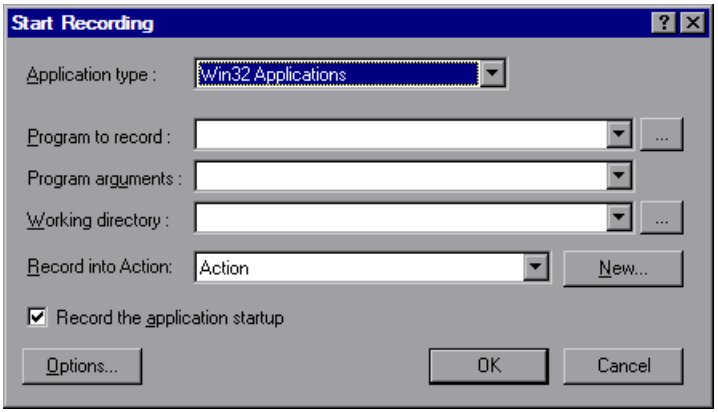


- **Program to record.** Select the browser or Internet application to record.

- **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:

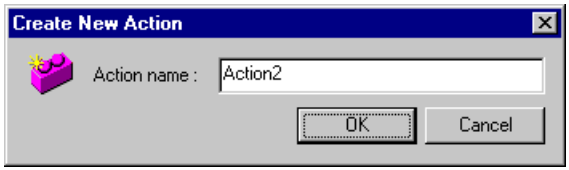


➤ **Program to record.** Enter the Win 32 application to record.

➤ **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:

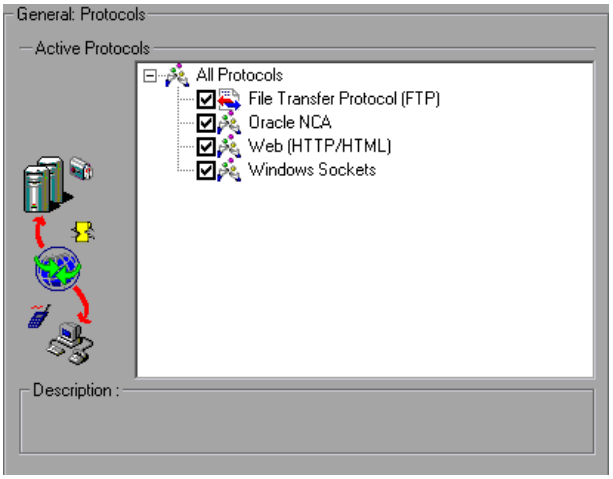
- If you are recording multiple actions, in which case you only need to perform the startup in one action.
- In cases where you want to navigate to a specific point in the application before starting to record.
- If you are recording into an existing script.

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 select a language for code generation and to set the scripting options, click the **Script** tab. For details, see Chapter 76, "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 78, "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, select **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.

Select 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 99.

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

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

Enter an email address and send your email.

Viewing the Recording Logs

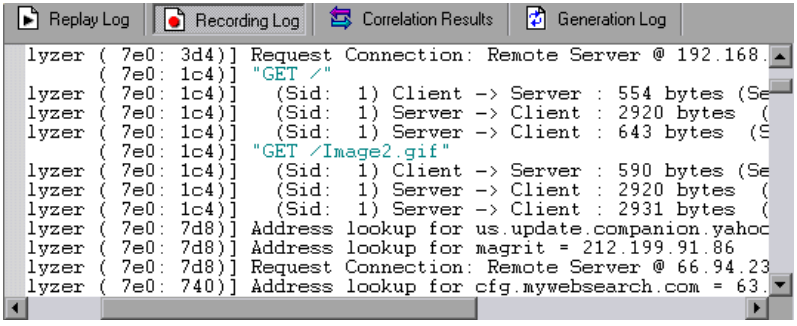
After recording, you can view the contents of the *vuser_init*, *Actions*, and *vuser_end* sections in the VuGen script editor. To display an action, select the action name in the left pane.

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. To open the script folder, switch to Script view (**View > Script View**) and select **Open Script Directory** from the right-click menu.

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

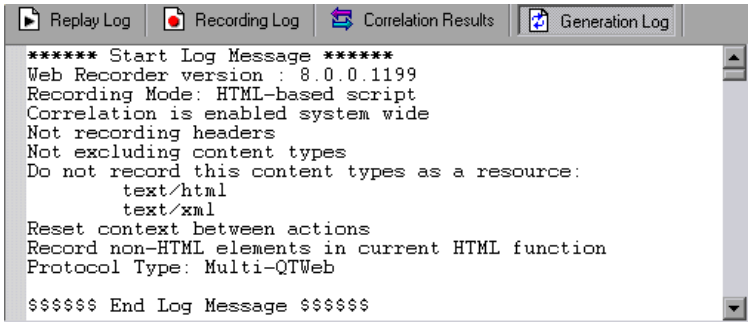
Recording Log

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, select **File > Zip Operations > Import from Zip File**. After you select 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, select **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, select **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 select 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, select **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.

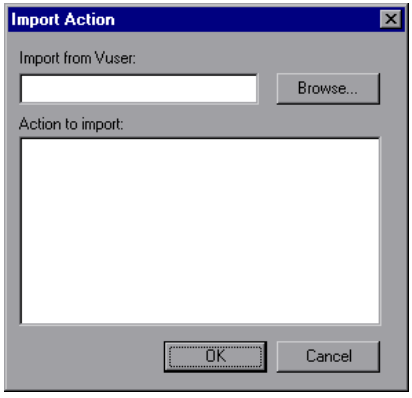
Importing Actions

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 79, "Configuring Run-Time Settings."

Providing Authentication Information

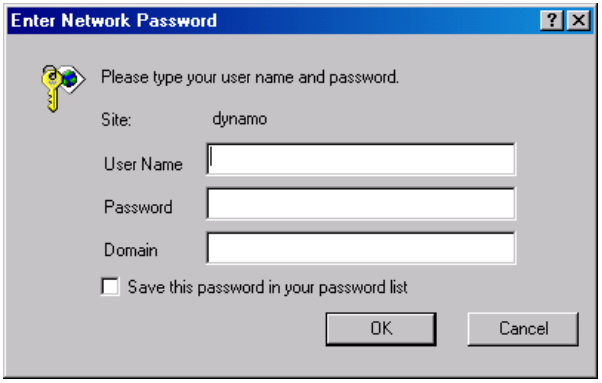
The following section only applies to multi-protocol recording.

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, right-click the password and select **Encrypt string**. VuGen encrypts the string and generates an **lr_decrypt** function, used to decode the password during replay. For more information about encrypting strings, see "Encrypting Text" on page 134.

Regenerating a Vuser Script

After recording a script, you can enhance it by adding transactions, rendezvous, messages, or comments. For more information, see Chapter 6, "Enhancing Vuser Scripts."

In addition, you can parameterize the script and correlate variables. For more information, see Chapter 70, "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.

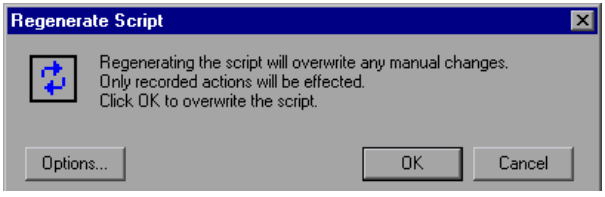
The following buttons are available from the Regenerate Script dialog box.

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.

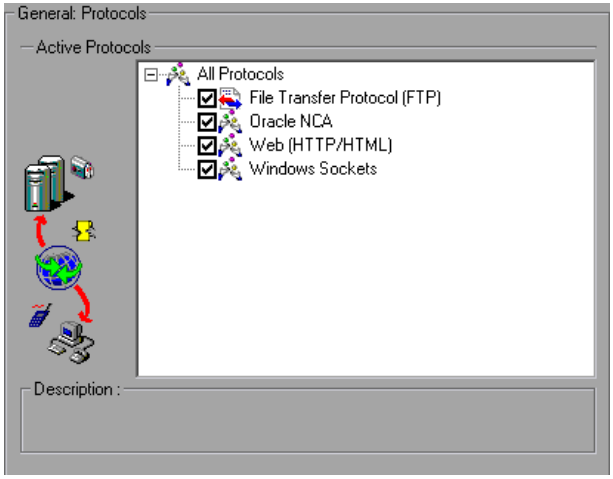
To regenerate a multi-protocol Vuser script:

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

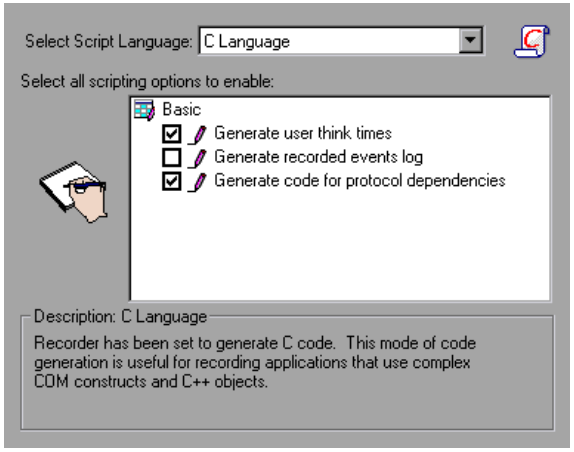


2 Click **Options** to open the Regenerate Options dialog box.

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.



6

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.

This chapter includes:

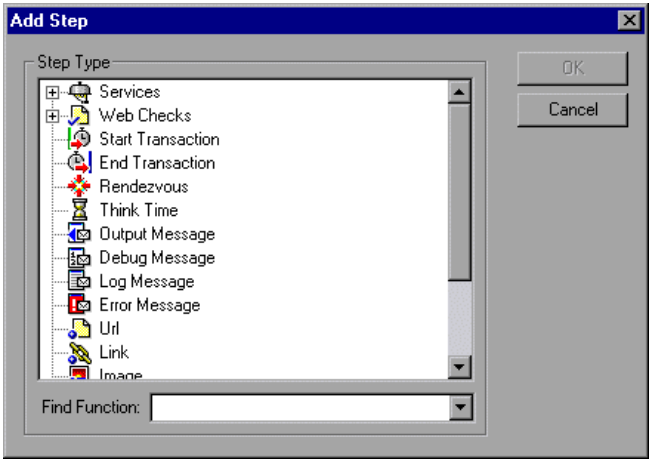
About Enhancing Vuser Scripts

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 Select **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, select **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, see 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, see 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 "Guidelines for Using C Functions" on page 641.

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 82. Alternatively, use the **Insert** menu to add **Start Transaction** and **End Transaction** markers.

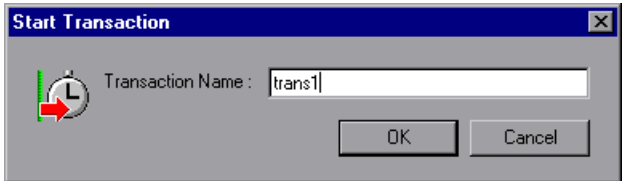
The following sections describe how to create a transaction during recording.

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)

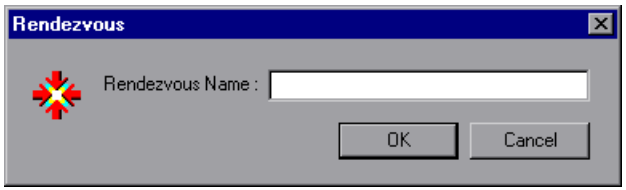
When performing load testing, you need to emulate heavy user load on your system. To accomplish this, you synchronize Vusers to perform a task at exactly the same moment. You configure multiple Vusers to act simultaneously by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point,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

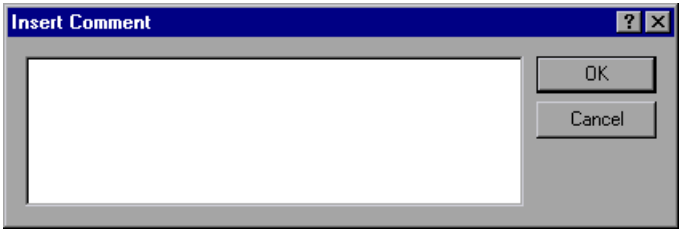
3 To insert rendezvous points into your script after the recording session, select **Insert** > **Rendezvous** from the VuGen menu.

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:

Function	Description
lr_get_attrib_string	Returns a command line parameter string.
lr_get_host_name	Returns the name of the machine running the Vuser script.
lr_get_master_host_name	Returns the name of the machine running the Controller. Not applicable when working with the HP Business 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, see 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, and to the Test Report summary. For example, you could insert a message that displays the current state of the client application. The LoadRunner Controller displays these messages in the Output window. You can also save these messages to a file.

When working with HP Business 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:

Funtion	Description
lr_debug_message	Sends a debug message to the Output window or the Business Process Monitor log file.
lr_error_message	Sends an error message to the Output window, Test Results report, or the Business Process Monitor log files.
lr_get_debug_message	Retrieves the current message class.
lr_log_message	Sends an output message directly to the log file, *output.txt*, located in the Vuser script 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, Test Results report, or the Business Process Monitor log files.

Funtion	Description
lr_set_debug_message	Sets a message class for output messages.
lr_vuser_status_messag e	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.

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

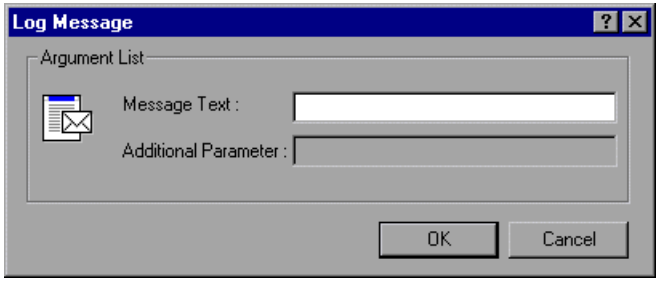
Using the **lr_output_message**, and **lr_error_message** functions, you can also send meaningful messages to the Test Results summary report. For information, see Chapter 10, "Viewing Test Results."

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.

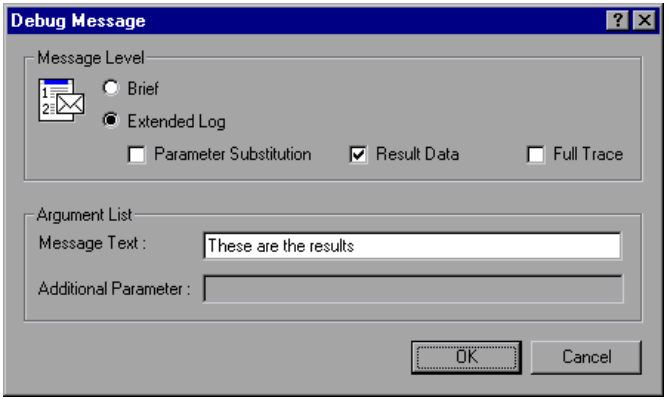
Debug Messages

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 select 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.

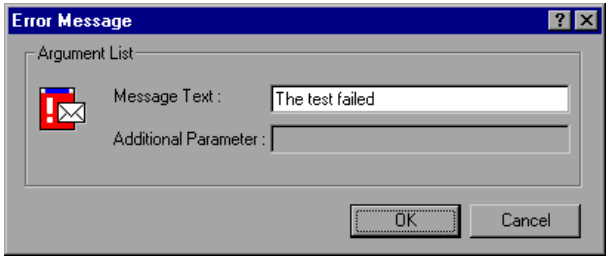
5 Click **OK** to insert the message and close the dialog box. An **lr_debug_message** function is inserted at the current point in the script.

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, see 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 1269.
- 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:

Function	Description
TE_wait_cursor	Waits for the cursor to appear at a specified location in the terminal window.
TE_wait_silent	Waits for the client application to be silent for a specified number of seconds.
TE_wait_sync	Waits for the system to return from X-SYSTEM or Input Inhibited mode.
TE_wait_text	Waits for a string to appear in a designated location.
TE_wait_sync_transaction	Records the time that the system remained in the most recent X SYSTEM mode.

For details on using synchronization functions in RTE Vuser scripts, see Chapter 64, "RTE - Synchronization."

Emulating User Think Time

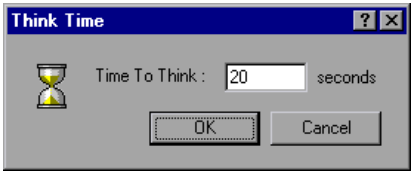
The time that a user waits between performing successive actions is known as the *think time*. Vusers use the **lr_think_time** function to emulate user think time. When your record a Vuser script, VuGen records the actual think times and inserts appropriate **lr_think_time** statements into the Vuser 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 Select **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, see 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 1267.

There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

Function	Description
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, see 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. Select **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. Select **View** > **Script View**.
2. Select the string you want to restore.
3. Select **Restore encrypted string (***string***)** from the right-click menu.

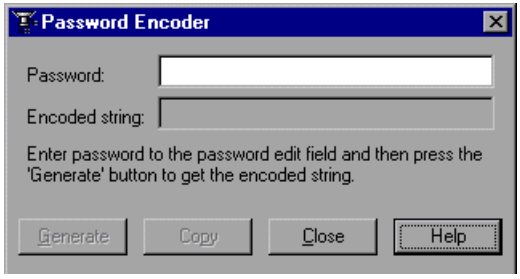
For more information on the **lr_decrypt** function, see the *Online Function Reference* (**Help** > **Function Reference**).

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 protect the integrity of the passwords. The **Password Encoder** enables you to encode your passwords and place secure values into the table.

To encode a password:

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.

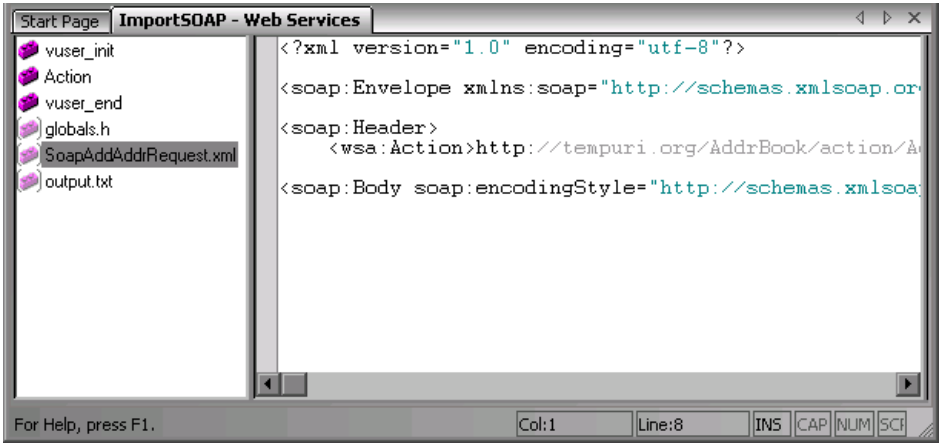
6 Click **Close** to close the Password Encoder.

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. Select **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.



7

Creating Business Process Reports

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

This chapter includes:

The following information applies to 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 Protocols.

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.

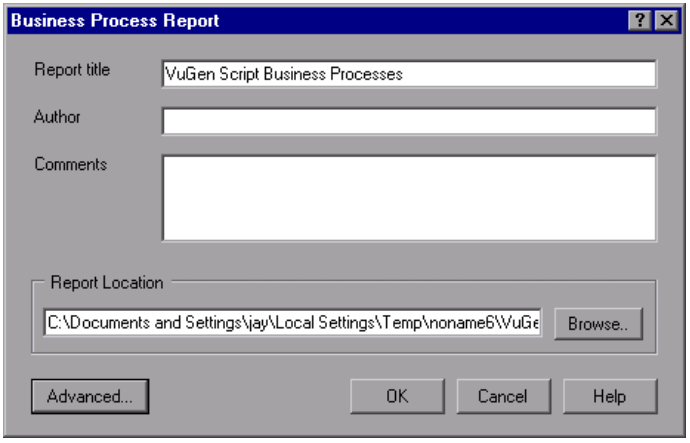
VuGen lets you customize the contents of the report by indicating what type of information you want to include.

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 Select **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.

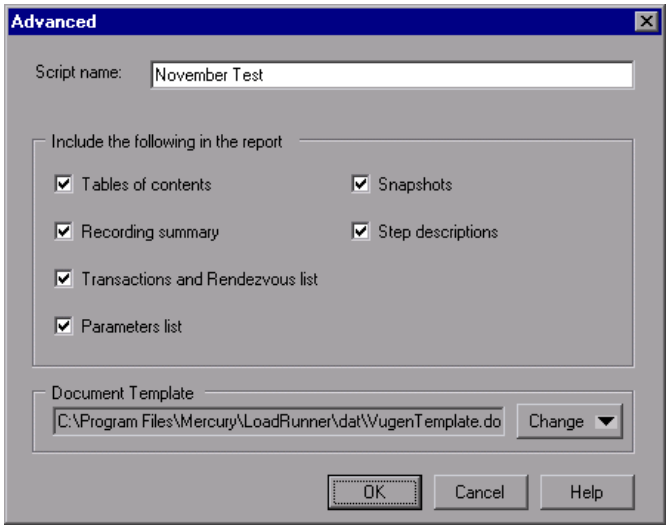
5 Accept the default report location and file name, or browse for the desired path. The default location is that of the saved script, and the default report name is **<script name>_Business_Processes.doc**

6 Click **Advanced** to specify the report's content. For more information about selecting content, see "Specifying Report Content" on page 139.

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, we recommend that you use an existing template as a basis for the new one. This will make sure that the required bookmarks and styles are maintained within the new template.

8

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.

This chapter includes:

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 145.

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**. See 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 Chapter 27, "Java - Correlating."
- For Database Vusers, see Chapter 34, "Database - Script Correlation."
- For Web Vusers, see Chapter 53, "Web (HTTP/HTML) Correlation Rules."
- For COM Vusers, see Chapter 40, "COM - Understanding and Correlating."

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 Vusers, see "Using Correlation Functions for Java Vusers" on page 144.

lr_eval_string	Replaces all occurrences of a parameter with its current value.
lr_save_string	Saves a null-terminated string to a parameter.
lr_save_var	Saves a variable length string to a parameter.

For additional information about the syntax of these functions, see the *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 Vusers, you can use the Java Vuser correlation functions. These functions may be used for all Java type Vusers, to save a string to a parameter and retrieve it when required.

lr.eval_string	Replaces a parameter with its current value.
lr.eval_data	Replaces a parameter with a byte value.
lr.eval_int	Replaces a parameter with an integer value.

lr.eval_string	Replaces a parameter with a string.
lr.save_data	Saves a byte as a parameter.
lr.save_int	Saves an integer as a parameter.
lr.save_string	Saves a null-terminated string to a parameter.

When recording a CORBA or RMI session, VuGen performs correlation internally. For more information, see Chapter 27, "Java - Correlating."

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

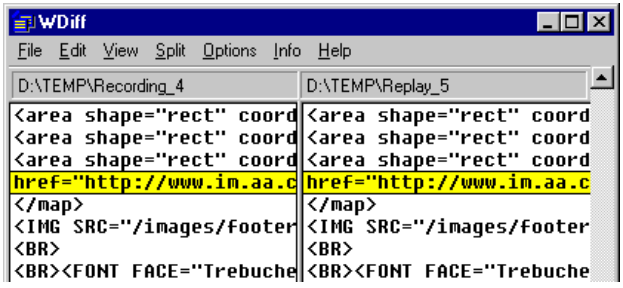
A useful tool in determining which values to correlate is *WDiff*. This utility lets you compare recorded scripts and results to determine which values need to be correlated.

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.

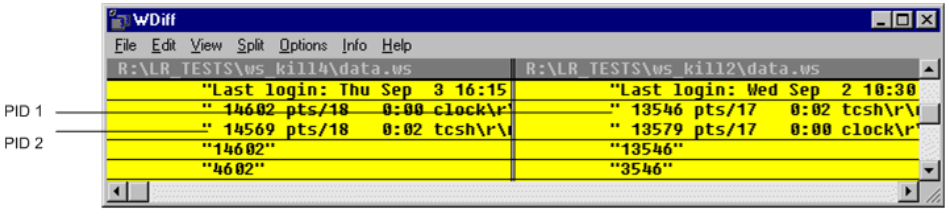
To use *WDiff* effectively, you record the identical operation twice, and compare the scripts (or data files for Tuxedo, WinSock, and Jolt). WDiff displays differences in yellow. Note that not all differences indicate a value to correlate. For example, certain receive buffers that indicate the time of 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 47

To continue with correlation, see the appropriate section:

- For Java language Vusers, see Chapter 27, "Java - Correlating."
- For Database Vusers, see Chapter 34, "Database - Script Correlation."
- For Web Vusers, see Chapter 53, "Web (HTTP/HTML) Correlation Rules."
- For COM Vusers, see Chapter 40, "COM - Understanding and Correlating."
- For Tuxedo Vusers, see Chapter 67, "Tuxedo Protocols."
- For WinSock Vusers, see Chapter 36, "Windows Sockets (WinSock) Protocol."

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"));
```

9

Running Vuser Scripts in Standalone 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.

This chapter includes:

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, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availability Center User Guides*.

Before you run a script in standalone mode, you can:

- enhance the script with Vuser functions (see Chapter 6, "Enhancing Vuser Scripts" or the *Online Function Reference*).
- parameterize the script (see Chapter 70, "Working with VuGen Parameters")
- correlate queries in the script (see Chapter 8, "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 from the Unix Command Line" on page 1389.

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.

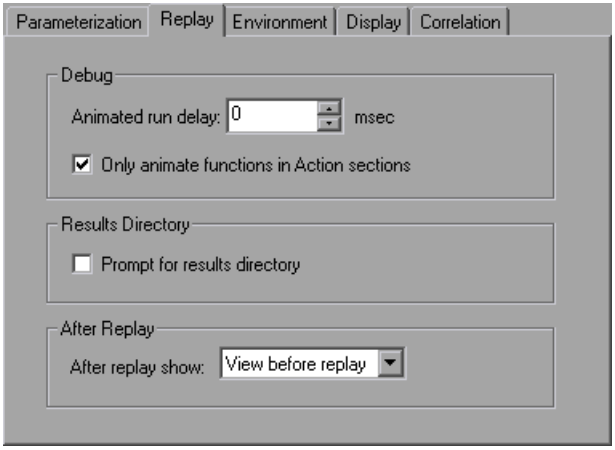
Prompt for results directory becomes disabled.

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.

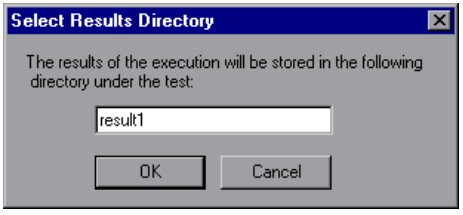


3 Select the **Replay** tab.

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.

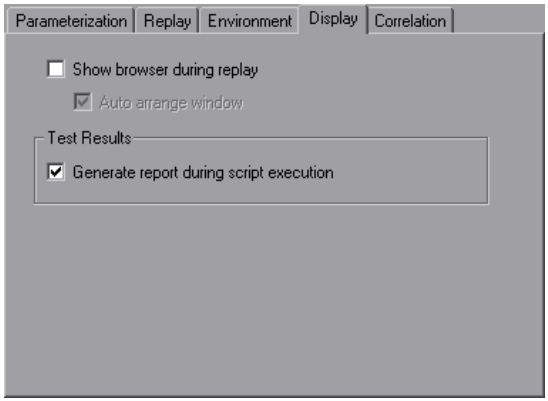
6 Select **Prompt for results directory** to be prompted for a results directory before running a script from VuGen. The Select Results Directory dialog box opens when you click the run command.



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 162.

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 155.

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 156.

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.

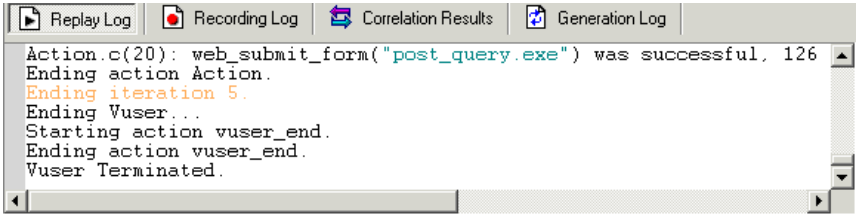
Various colors of text are used in the Replay Log.

- **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.

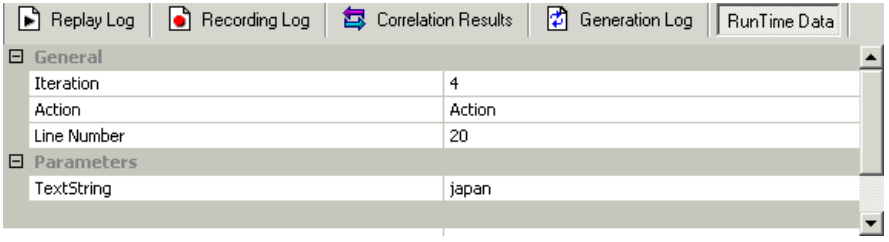
For more information on closing and opening the Output window, see "Replaying a Vuser Script" on page 154.

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 70, "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 162.

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.

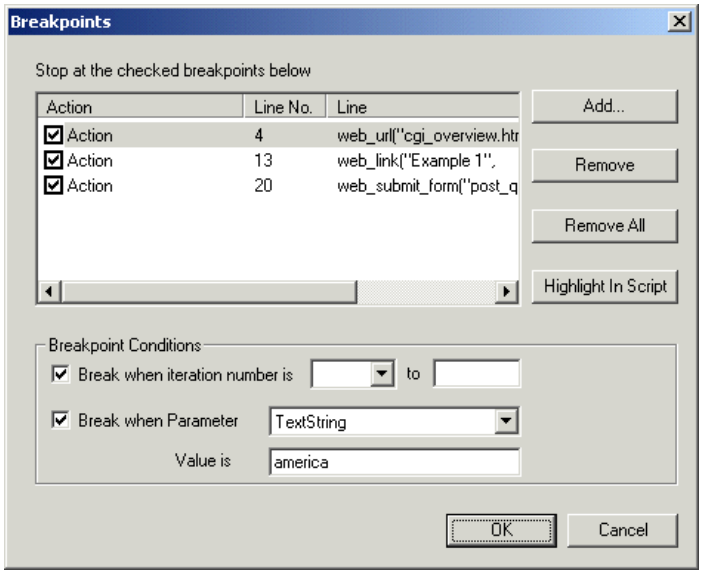
2 To resume execution, select **Vuser** > **Run**.

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:

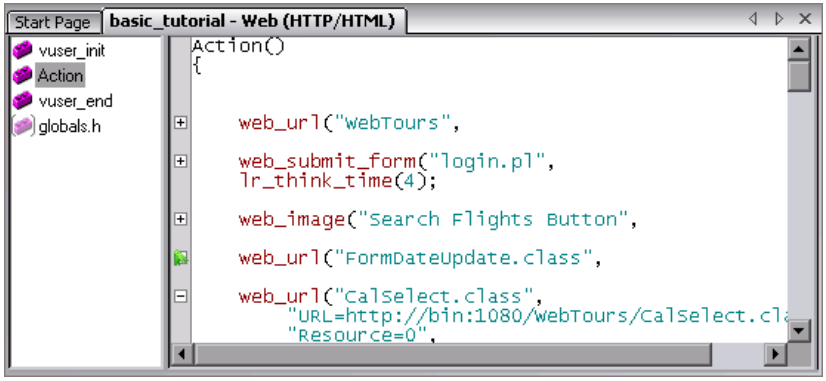
1. To pause the script after a specific number of iterations, select **Break when iteration number is** and enter the desired number.
2. To pause the script when parameter *X* has a specific value, select **Break when Parameter** *X* **Value is** and enter the desired value. For more information about parameters, see Chapter 70, "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. Select **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 select **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, select **View > Test Results** and click F1 to open the online help.

For information on setting the above Display options, see "Setting the Display Options" on page 153.

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 155.
- **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 56.
- **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.

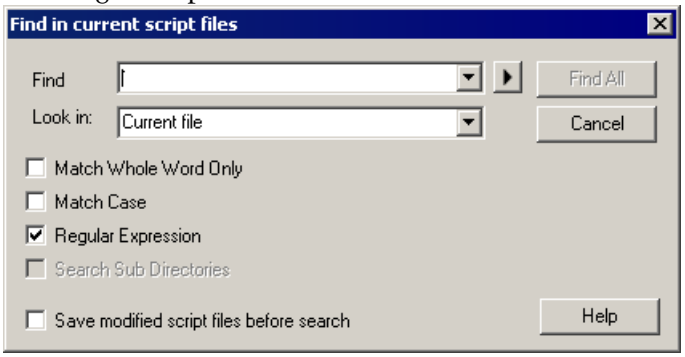
Vugen displays all matches in a separate results window. You double-click on any line in the window to open the relevant file.

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.

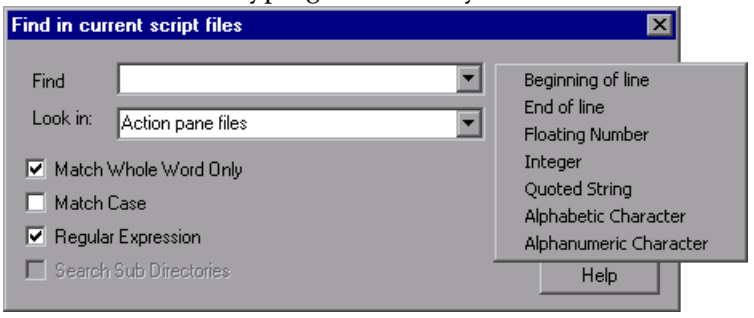
2 Select **Edit** > **Find in current script files.** The Find in current script files dialog box opens.



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: We recommend 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 166.

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, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

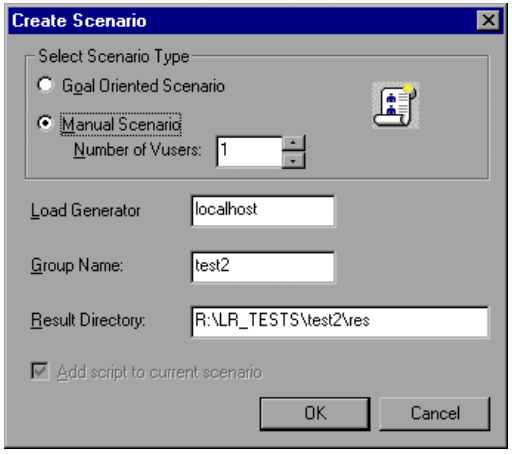
Using VuGen to Create a LoadRunner Scenario

Note: The following section only applies to LoadRunner. For information on integrating scripts into Business Process profiles, see 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 Select **Tools** > **Create Controller Scenario**. The Create Scenario dialog box opens.



2 Select 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, see the *HP LoadRunner Controller User's Guide*.

10

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.

This chapter includes:

About Viewing Test Results

After you run a script, the Test Results window displays all aspects of the test run and can include:

- a high-level results overview report (pass/fail status)
- the data used in all runs
- an expandable tree of the steps, specifying exactly where application failures occurred
- the exact locations in the script where failures occurred
- a still image of the state of your application at a particular step
- a movie clip of the state of your application at a particular step or of the entire test
- detailed explanations of each step and checkpoint pass or failure, at each stage of the test

Reports for Web Services Vusers contain several enhacements, such as breakdown by operations, checkpoint results, and a view of the HTTP traffic. For more information, see "Viewing Web Services Reports" on page 193.

You can open the Test Results window as a standalone application from the **Start** menu. To open the Test Results window, choose **Start** > **All Programs** > **HP Service Test** > **Test Results Viewer**.

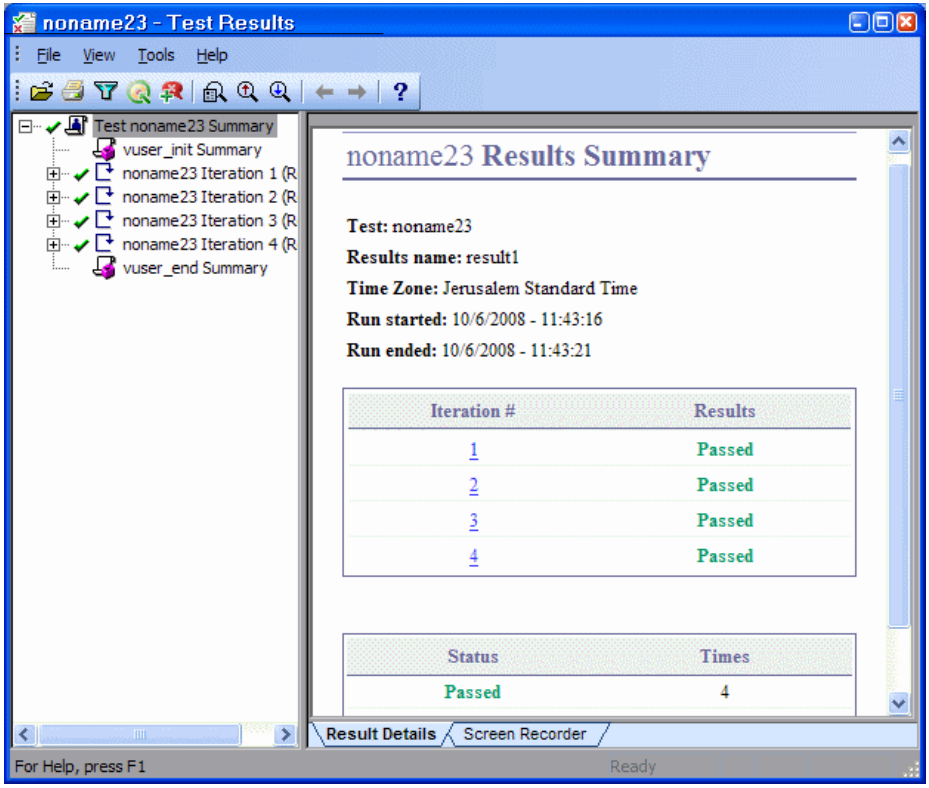
Note: The Test Results window can show results with up to 300 levels in the tree hierarchy. If you have results with more than 300 nested levels, you can view the entire report by manually opening the **results.xml** file.

The Test Results Window

After a run session, you view the results in the Test Results window.

- 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.



The Test Results window contains the following key elements:

- **Test results title bar.** Displays the name of the test.
- **Menu bar.** Displays menus of available commands.
- **Run results toolbar.** Contains buttons for viewing test results (choose **View** > **Test Results Toolbar** to display the toolbar). For more information, see "Test Results Toolbar" on page 175.
- **Run results tree.** Contains a graphic representation of the test results in the run results tree. The run results tree is located in the left pane of the Test Results window. For more information, see "Run Results Tree" on page 174.
- **Result Details tab.** Contains details of the selected node in the run results tree. The Result Details tab is located in the right pane of the Test Results window. For more information, see "Run Results Details" on page 175.
- **Screen Recorder tab.** Contains the recorded movie associated with the test results. The screen recorder tab is located in the right pane of the Test Results window. For more information, see "Capturing and Viewing Still Images and Movies of Your Application" on page 183.
- **Status bar.** Displays the status of the currently selected command (choose **View** > **Status Bar** to view the status bar).

You can change the appearance of the Test Results window. For more information, see "Changing the Appearance of the Test Results Window" on page 176.

Run Results Tree

The left pane in the Test Results window displays the **run results tree**—a graphical representation of the test results:


- indicates a step that succeeded.


- indicates a step that failed.

- indicates a warning, meaning that the step did not succeed, but it did not cause the test to fail.


- indicates a step that failed unexpectedly.

You can collapse or expand a branch in the run results tree to change the level of detail that the tree displays.

Run Results Details

By default, when the Test Results window opens, it displays a summary of the script run in the **Result Details** tab in the right pane of the window.

The right pane of the Test Results Window contains tabs labeled **Result Details** and **Screen Recorder**. When you select the top node of the run results tree, the Result Details tab shows a summary of the results for your test. When you select a branch or step in the tree, the Result Details tab contains the details for that step. The Result Details tab may also include a still image of your application for the highlighted step.

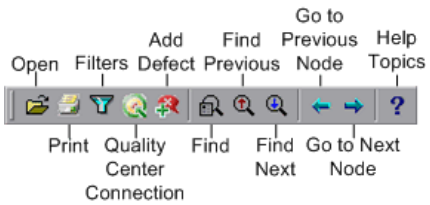
The Screen Recorder tab contains the movie associated with your test results. If there is no movie associated with your test results, the Screen Recorder tab contains the message: No movie is associated with the results.

For more information on viewing snapshots and movies of your application, see "Capturing and Viewing Still Images and Movies of Your Application" on page 183.

When you select the top node of the results tree in the **Result Details** tab, it indicates the script name, results name, the start and end date and time of the run, the number of iterations, and whether an iteration passed or failed. For checkpoints in Web Services scripts, the possible results are **Passed** or **Failed**. If an iteration does not contain checkpoints, the possible results are **Done** or **Failed**.

Test Results Toolbar

The Test Results toolbar contains buttons for viewing test results.



Changing the Appearance of the Test Results Window

By default, the Test Results window has the same look and feel as the QuickTest window, using the Microsoft Office 2003 theme. You can change the look and feel of the Test Results window, as required.

To change the appearance of the Test Results window:

In the Tests Results window, choose **View > Window Theme**, and then select the way the window should appear from the list of available themes. For example, you can apply a Microsoft Office 2000 or Microsoft Windows XP theme.

Note: You can apply the Microsoft Windows XP theme to the Tests Results window only if your computer is set to use a Windows XP theme.

Viewing the Results

By default, VuGen generates test results and automatically opens them at the end of a run.

To prevent VuGen from generating the results, choose **Tools** > **General Options**, select the **Display** tab and clear the **Generate report during script execution option**.

To indicate whether or not to open the results after running the script, choose **Tools** > **General Options**, select the **Replay** tab, and select a view in the **After Replay** section.

For more information on setting the Display options, see "Running Vuser Scripts in Standalone Mode" in the *Virtual User Generator* User Guide.

In addition, you can view the results of previous runs of the current script, and results of other script. You can also preview results on screen and print them to your default Windows printer, as well as export them to an HTML file.

For more information, see:

➤ "About Viewing Test Results" on page 172

➤ "The Test Results Window" on page 172

➤ "Viewing the Results" on page 176

➤ "Capturing and Viewing Still Images and Movies of Your Application" on page 183

➤ "Finding Results Steps" on page 186

➤ "Printing Results" on page 187

➤ "Previewing Test Results" on page 188

➤ "Exporting Test Results" on page 190

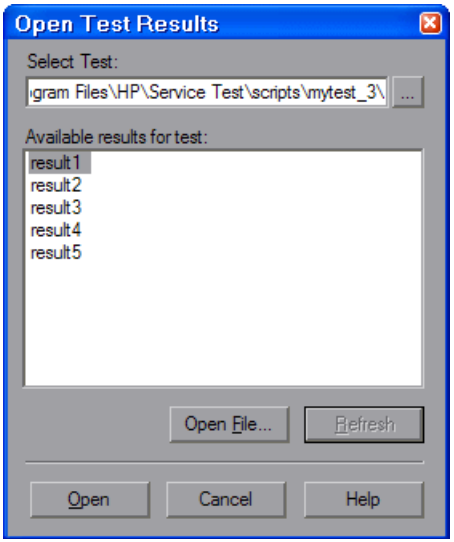
Opening Test Results to View a Selected Run

You can view the saved results for the current script, or you can view the saved results for other scripts.

To select a specific set of test results:

1 Choose **File** > **Open** from within the Test Results window.

2 To view all of the results, specify the parent script path.



3 Select a results set and click **Open**.

Tip: To update the results list after you specify a new path, click **Refresh**.

Searching for Results in the File System

By default, the results saved in the file system are stored in the script folder. If you are connected to Quality Center, the results are stored together with You can search for results in the file system by script or by result file.

To search for results in the file system by script:

1 In the Open Test Results dialog box, enter the path of the folder that contains the results file for your script, or click the browse button to open the Open Test dialog box.

2 Find and highlight the script whose results you want to view, and click **Open**.

3 In the Open Test Results dialog box, highlight the result set you want to view, and click **Open**. The Test Results window displays the selected results.

To search for results in the file system by result file:

1 In the Open Test Results dialog box, click the **Open File** button to open the Select Results File dialog box.

2 Browse to the folder where the results file is stored.

3 Highlight the results (**.xml**) file you want to view, and click **Open**. The Test Results window displays the selected results.

Notes:

➤ By default, result files for tests are stored in <**Script**>\<**ResultName**>.

➤ Results files for earlier versions were saved with a **.qtp** file extension. In the Select Results File dialog box, only results files with an **.xml** extension are shown by default. To view results files with a **.qtp** extension in the Select Results File dialog box, select **Test Results (*.qtp)** in the **Files of type** box.

Searching for Results Saved in Quality Center

If your script is stored in Quality Center, the results are also stored in Quality Center. You cannot change the location of the test results.

To search for test results saved in Quality Center:

1 In the Test Results window, choose **Tools** > **Quality Center Connection** or click the **Quality Center Connection** button and connect to your Quality Center project.

2 In the Open Test Results dialog box, enter the path of the folder that contains the results file for your QuickTest test, or click the browse button to open the Open Test from Quality Center Project dialog box.

3 Select **DB Vuser** in the **Test Type** list.

4 Find and highlight the script whose test results you want to view, and click **OK**.

5 In the Open Test Results dialog box, highlight the test result set you want to view, and click **Open**. The Test Results window displays the selected test results.

For more information on working with Quality Center, see Chapter 11, "Managing Scripts Using Quality Center".

Working in Test Results Tree

The following steps describe how to work within the nodes of the Test Results tree:

1 You can collapse or expand a branch in the run results Tree to select the level of detail that the tree displays.

- To collapse a branch, select it and click the collapse (–) sign to the left of the branch icon, or press the minus key (–) on your keyboard number pad. The details for the branch disappear in the results tree, and the collapse sign changes to expand (+).
- To collapse all of the branches in the run results tree, choose **View** > **Collapse All** or right click a branch and select **Collapse All**.
- To expand a branch, select it and click the expand (+) sign to the left of the branch icon, or press the plus key (+) on your keyboard number pad. The tree displays the details for the branch and the expand sign changes to collapse.

  If you just opened the Test Results window, the tree expands one level at a time. If the tree was previously expanded, it reverts to its former state.
- To expand a branch and all branches below it, select the branch and press the asterisk (*) key on your keyboard number pad.
- To expand all of the branches in the run results tree, choose **View** > **Expand All**; right click a branch and select **Expand All**; or select the top level of the tree and press the asterisk (*) key on your keyboard number pad.

2 You can view the results of an individual iteration, an action, or a step. When you select a step in the run results tree, the right side of the Test Results window contains the details of the selected step. Depending on your settings in the Run tab of the Options dialog box, the right side of the Test Results window may be split into two panes, with the bottom pane containing a still image (or in selected cases, other data) of the selected step. The right pane may also contain a movie of your application. For more information, see "Capturing and Viewing Still Images and Movies of Your Application" on page 183.

The results can be one of the following types:

- Iterations, actions, and steps that contain checkpoints are marked **Passed** or **Failed** in the right part of the Test Results window and are identified by icons in the tree pane.
- Iterations, actions, and steps that ran successfully, but do not contain checkpoints, are marked **Done** in the right part of the Test Results window.
- Steps that were not successful, but did not cause the script to stop running, are marked **Warning** in the right part of the Test Results window and are identified by the icon ! or !.

3 To filter the information displayed in the Test Results window, click the **Filters** button or choose **View** > **Filters**. The Filters dialog box opens.

The default filter options are displayed in the image above. The Filters dialog box contains the following options:

Iterations area:

- **All.** Displays test results from all iterations.
- **From iteration X to Y.** Displays the test results from a specified range of test iterations.

Status area:

- **Fail.** Displays the results for the steps that failed.
- **Warning.** Displays the results for the steps with the status **Warning** (steps that did not pass, but did not cause the script to fail).
- **Pass.** Displays the results for the steps that passed.
- **Done.** Displays the results for the steps with the status **Done** (steps that were performed successfully but did not receive a pass, fail, or warning status).

Content area:

- **All.** Displays all steps from all nodes in the test.
- **Show only actions.** Displays the action nodes in the test (not the specific steps in the action nodes).

4 To find specific steps within the Test Results, click the **Find** button or choose **Tools** > **Find**. For more information, see "Finding Results Steps."

5 To move between previously selected nodes within the run results tree, click the **Go to Previous Node** or **Go to Next Node** buttons.

6 To view the results of other run sessions, click the **Open** button or choose **File** > **Open.** For more information, see "Opening Test Results to View a Selected Run" on page 177.

7 To print results, click the **Print** button or choose **File** > **Print**. For more information, see "Printing Results" on page 187.
(You can preview the run results before you print them. For more information, see "Previewing Test Results" on page 188.)

Note: If you have Quality Center installed, you can add a defect to a Quality Center project. For more information, see "Submitting Defects to Quality Center" on page 191.

8 To export the run results to an HTML file, choose **File** > **Export to HTML File**. For more information, see "Exporting Test Results" on page 190.

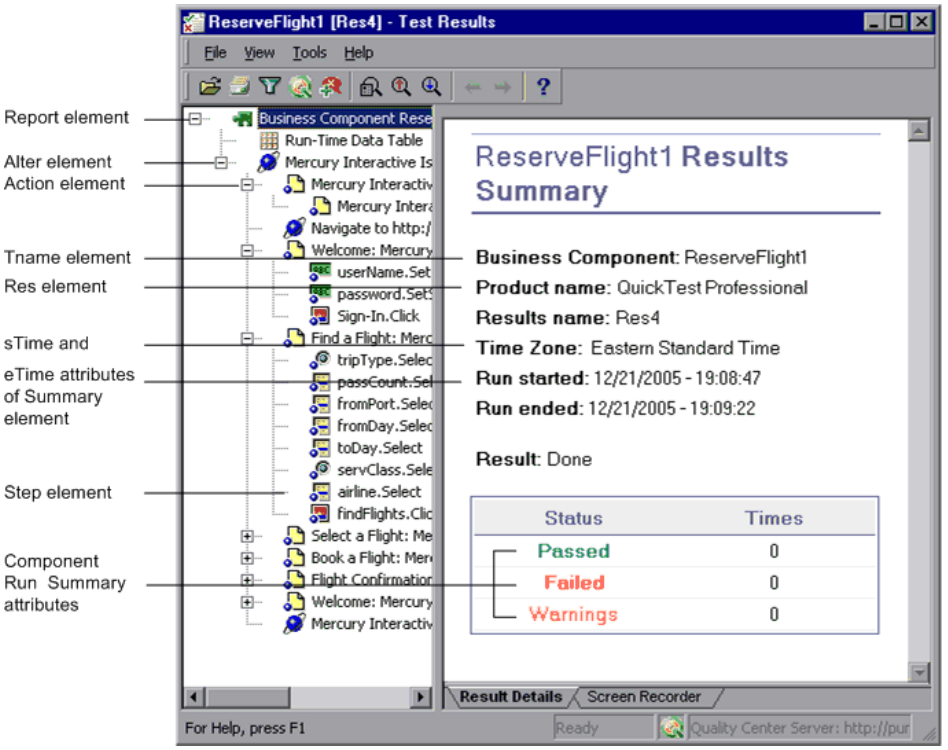
9 Choose **File** > **Exit** to close the Test Results window.

Capturing and Viewing Still Images and Movies of Your Application

The Test Results Viewer can capture still images and movies of your application. These captured files can be viewed in the Test Results window. The right pane of the Test Results window contains tabs labeled **Result Details** and **Screen Recorder**. These tabs enable you to view either still images and text details, or a movie of your application.

Viewing Still Images of Your Application

By default, the Test Results viewer saves a still image of your application for failed steps. When you select a failed step in the results tree and select the **Result Details** tab, the bottom right pane of the Test Results window displays a screen capture of your application corresponding to the highlighted step in the run results tree.

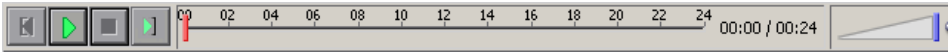


If the highlighted step does not contain an error, the right pane contains the result details with no screen capture.

Viewing Movies of Your Run Session

the Test Results viewer can save a movie of your application during a run session. This can be useful to help you see how your application behaved under test conditions or to debug your script. You can view the entire movie or select a particular segment to view. When you select a step in the run results tree and click the **Screen Recorder** tab, the right pane of the Test Results window displays the frame in the movie corresponding to the highlighted step in the run results tree.

The top of the Screen Recorder tab contains controls that enable you to play, pause, stop, jump to the first frame of the movie, jump to the last frame of the movie, and control the volume. You can also drag the slider bar to scroll through the movie.



Tip: Double-click the right pane of the Test Results window to expand the Screen Recorder and hide the run results tree. Double-clicking again restores the Screen Recorder to its previous size and displays the run results tree. When the Screen Recorder is expanded, the playback controls at the top of the Screen Recorder automatically hides after approximately three seconds with no mouse activity, or when you click anywhere on the Screen Recorder. They reappear when you move the mouse again.

Removing a Movie from the Test Results

You can remove a stored movie from the results. This reduces the size of the results file. To remove a movie from the results, choose **File** > **Remove Movie from Results**.

Exporting Captured Movie Files

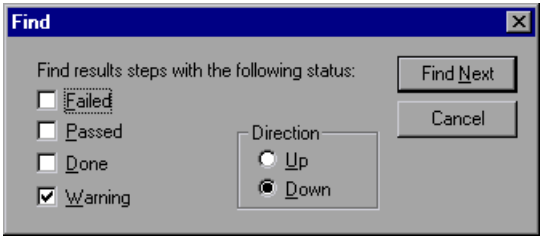
You can export a captured Screen Recorder movie to a file. The file is saved as an **.fbr** file. You can view **.fbr** files in the HP Micro Recorder. You can also attach **.fbr** files to defects in Quality Center. Quality Center users who have the QuickTest Add-in for Quality Center installed can view the movies from Quality Center.

To export a Screen Recorder movie:

1. Choose **File** > **Export Movie to File**. The Save As dialog box opens, enabling you to change the default destination folder and rename the file, if required. By default, the file is saved in the script results folder.
2. Click **Save** to save the exported (**.fbr**) file and close the dialog box.

Finding Results Steps

The Find dialog box enables you to find specified steps such as errors or warnings from within the Test Results. You can select a combination of statuses to find, for example steps that are both **Passed** and **Done**.



The following options are available:

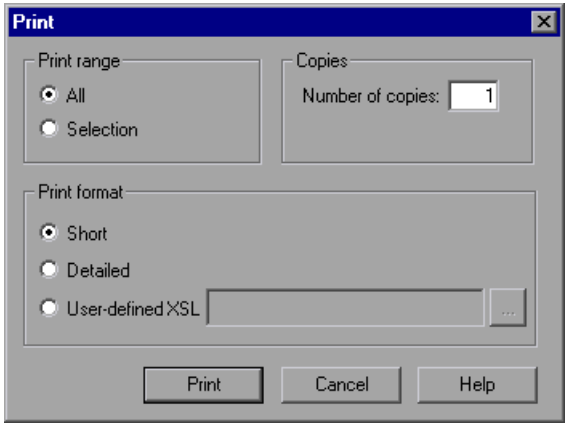
Option	Description
Failed	Finds a failed step in the Test Results.
Warning	Find a step where a warning was issued.
Passed	Finds a passed step in the Test Results.
Done	Finds a step that has finished its run.
Direction	Indicates whether to search up or down within the steps of the Test Results.

Printing Results

You can print results from the Test Results window. You can select the type of report you want to print, and you can also create and print a customized report.

To print the results:

1 Click the **Print** button or choose **File** > **Print**. The Print dialog box opens.



2 Select a **Print range** option:

- **All.** Prints the results for the entire script.
- **Selection.** Prints the results for the selected branch in the run results tree.

3 Specify the **Number of copies** of the results that you want to print.

4 Select a **Print format** option:

- **Short.** Prints a summary line (when available) for each item in the run results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Prints all available information for each item in the run results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 193.

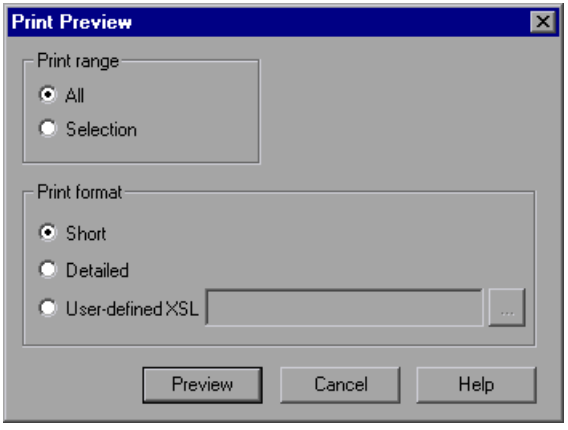
5 Click **Print** to print the selected results information to your default Windows printer.

Previewing Test Results

You can preview results on screen before you print them. You can select the type and quantity of information you want to view, and you can also display the information in a customized format.

To preview the results:

1 Choose **File** > **Print Preview**. The Print Preview dialog box opens.



2 Select a **Print range** option:

- **All.** Previews the results for the entire script.
- **Selection.** Previews results information for the selected branch in the results tree.

3 Select a **Print format** option:

- **Short.** Previews a summary line (when available) for each item in the results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Previews all available information for each item in the results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the preview, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 193.

4 Click **Preview** to preview the appearance of your results on screen.

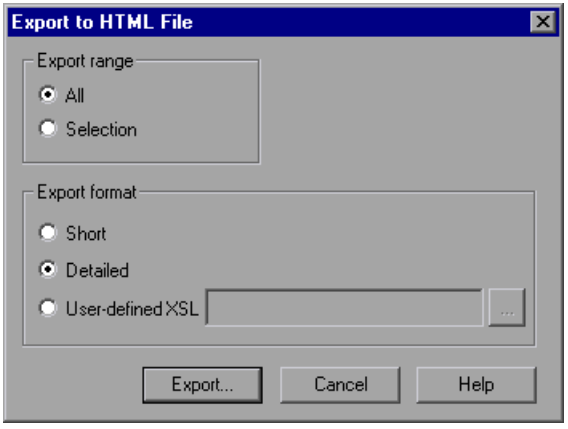
Tip: If some of the information is cut off in the preview, for example, if checkpoint names are too long to fit in the display, click the **Page Setup** button in the Print Preview window and change the page orientation from **Portrait** to **Landscape**.

Exporting Test Results

You can export the results to an HTML file. This enables you to view the results even if the Test Results Viewer environment is unavailable. For example, you can send the file containing the results in an e-mail to a third-party. You can select the type of report you want to export, and you can also create and export a customized report.

To export the results:

1 Choose **File** > **Export to HTML File**. The Export to HTML File dialog box opens.



2 Select an **Export range** option:

- **All.** Exports the results for the entire script.
- **Selection.** Exports result information for the selected branch in the results tree.

3 Select an **Export format** option:

- **Short.** Exports a summary line (when available) for each item in the results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Exports all available information for each item in the results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the exported report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 193.

4 Click **Export**. The Save As dialog box opens, enabling you to change the default destination folder and rename the file, if required. By default, the file is saved in the results folder.

5 Click **Save** to save the HTML file and close the dialog box.

Submitting Defects to Quality Center

When viewing the results, you can submit any defects detected to a Quality Center project directly from the Test Results window.

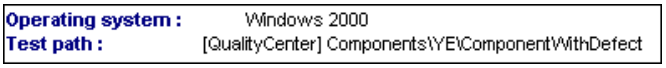
To manually submit a defect to Quality Center:

1 Ensure that the Quality Center client is installed on your computer. (Enter the Quality Center Server URL in a browser and ensure that the Login screen is displayed.)

2 Choose **Tools** > **Quality Center Connection** or click the **Quality Center Connection** button to connect to a Quality Center project. For more information on connecting to a project, see "Connecting to and Disconnecting from Quality Center" on page 200.

3 Choose **Tools** > **Add Defect** or click the **Add Defect** button to open the Add Defect dialog box in the specified Quality Center project. The Add Defect dialog box opens.

4 You can modify the defect information if required. Basic information on the is included in the description:



5 Click **Submit** to add the defect information to the Quality Center project.

6 Click **Close** to close the Add Defect dialog box.

Connecting to Quality Center from the Test Results Window

To manually submit defects to Quality Center from the Test Results window, you must be connected to Quality Center.

The connection process has two stages. First, you connect to a local or remote Quality Center server. This server handles the connections between the Test Results and the Quality Center project.

Next, you log in and choose the project you want to access. The project stores tests and run session information for the application you are testing. Note that Quality Center projects are password protected, so you must provide a user name and a password.

For more information on connecting to a Quality Center project, see "Connecting to and Disconnecting from Quality Center" on page 200.

Customizing the Test Results Display

Each result set is saved in a single **.xml** file (called **results.xml**). This **.xml** file stores information on each of the test result nodes in the display. The information in these nodes is used to dynamically create **.htm** files that are shown in the top-right pane of the Test Results window.

Each node in the run results tree is an element in the **results.xml** file. In addition, there are different elements that represent different types of information displayed in the test results. You can take test result information from the **.xml** file and use XSL to display the information you require in a customized format (either when printing from within the Test Results window, when displaying test results in your own customized results viewer, or when exporting the test results to an HTML file).

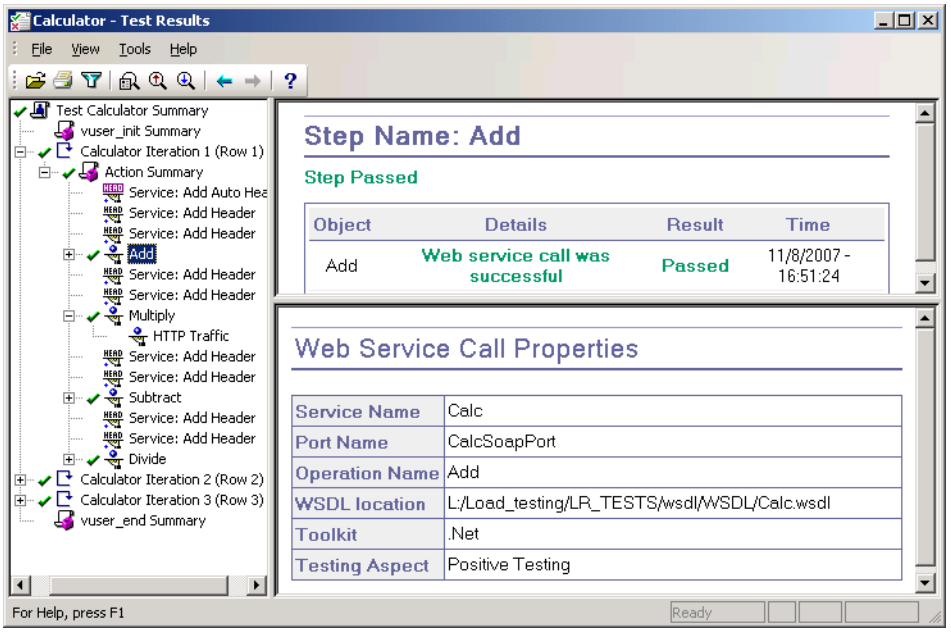
XSL provides you with the tools to describe exactly which test result information to display and exactly where and how to display, print or export it. Using a XSL editor, you can modify the **.css** and **.xsl** files in the results folder, to change the appearance of the report (for example, fonts, colors, and so forth).

For example, in the **results.xml** file, one element tag contains the name of an action, and another element tag contains information on the time at which the run was performed. Using XSL, you could tell your customized editor that the action name should be displayed in a specific place on the page and in a bold green font, and that the time information should not be displayed at all.

Viewing Web Services Reports

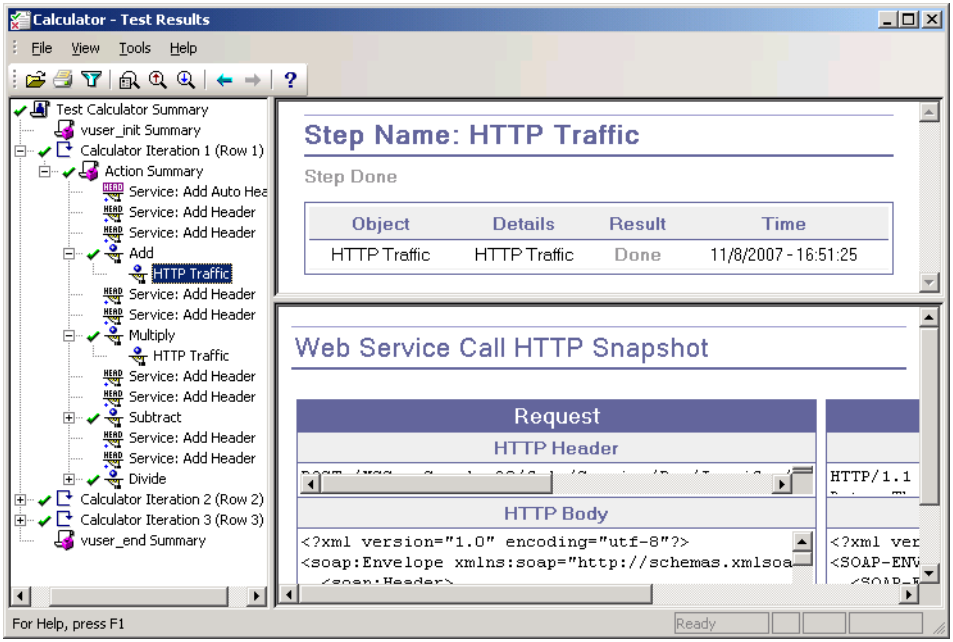
This section describes the report information specific to Web Services scripts.

The Web Services test results shows a list of the service's operations that were called during replay. When you select an operation, the report shows information about the service, operation, toolkit, testing aspect, and WSDL.



If VuGen cannot interpret the script or if it encounters another type of error, the report displays a message in the right pane stating the problem.

If you expand an operation's node further, the report shows the actual SOAP traffic for the Request and Response.

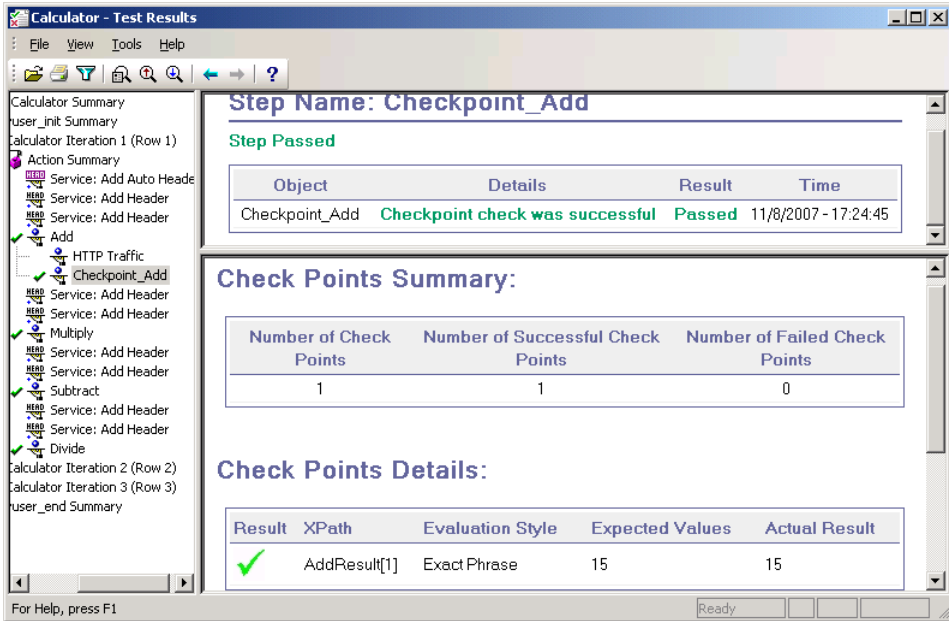


Checkpoint Results

The Results window also shows checkpoint results. Setting checkpoints is described in "Checkpoints" on page 330

If your checkpoints fail, the report provides a summary with a reason for the failure. It also provides the Expected Value and Actual Results as well as the argument tree.

To view the checkpoint details, expand the appropriate step under the operation in the left pane and click the **Checkpoint** node.



Sending Custom Information to the Report

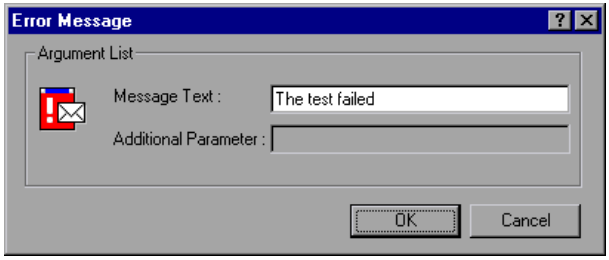
In addition to the information sent automatically to the report, for Web Service Vusers, you can send information to the report using the message functions **lr_output_message** or **lr_error_message**.

You can use the following message functions in your Vuser scripts:

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, and the messages will be sent to the Test Summary report.

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

11

Managing Scripts Using Quality Center

VuGen's integration with Quality Center lets you manage Vuser scripts using Quality Center.

This chapter includes:

About Managing Scripts Using Quality Center

VuGen 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 VuGen 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, see the *Quality Center User 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 VuGen and the Quality Center project.

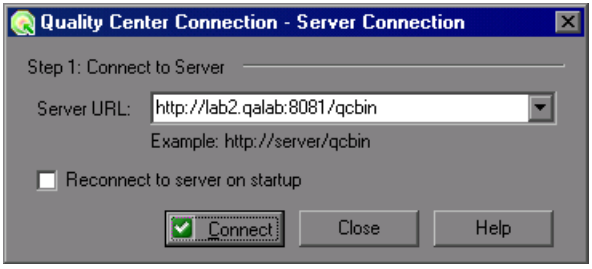
Next, you select the project you want to access. The project stores the scripts that you created in VuGen.

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.

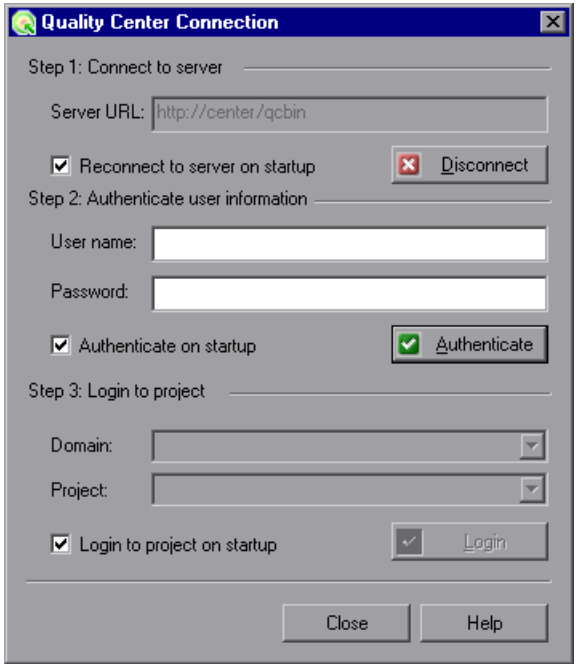


3 In the **Server URL** box, type the URL address of the Web server where Quality Center is installed.

Note: You can select 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 VuGen, 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 VuGen, 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.

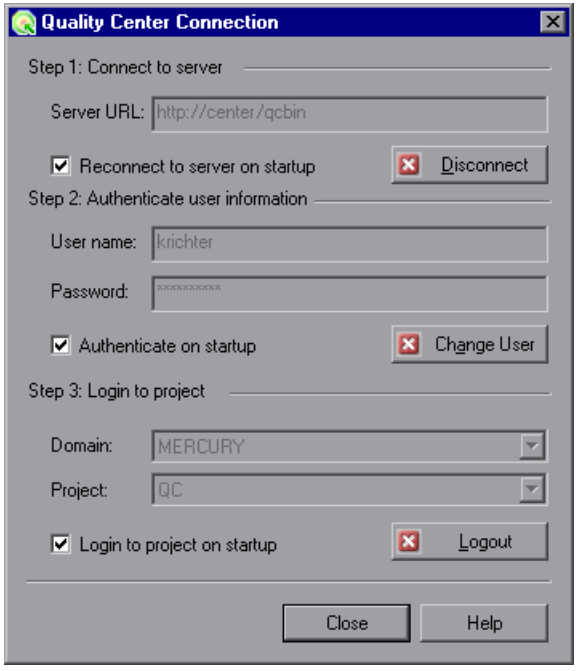
10 Click **Close** to close the Quality Center Connection dialog box.

Disconnecting from Quality Center

You can disconnect VuGen 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.

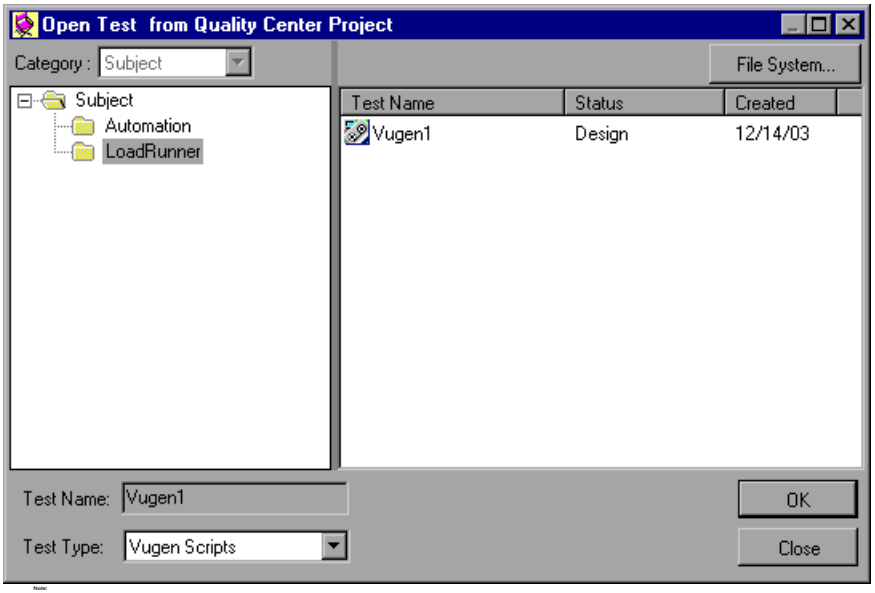
Opening Scripts from a Quality Center Project

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 Select **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. VuGen 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, VuGen 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.

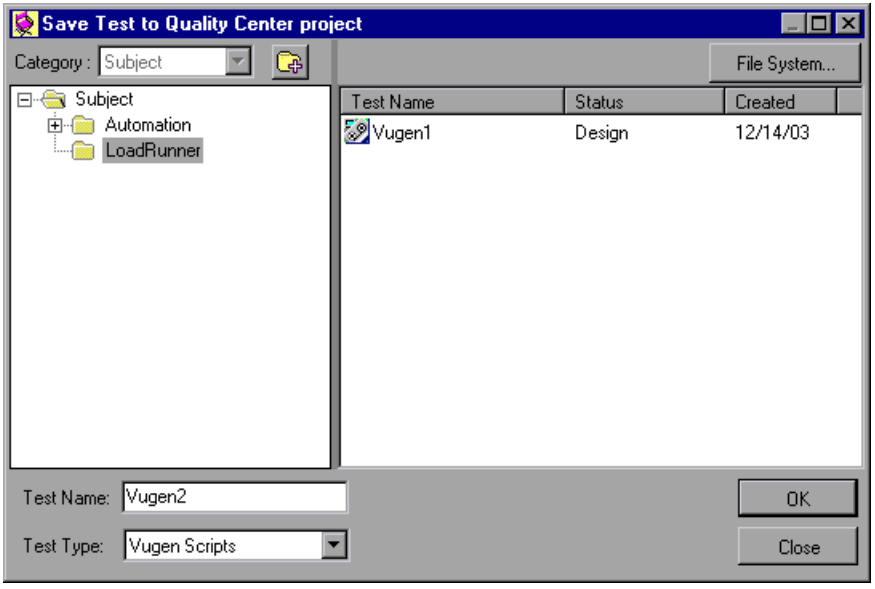
Saving Scripts to a Quality Center Project

When VuGen is connected to a Quality Center project, you can create new scripts in VuGen 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 Select **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 to 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.

This section includes:

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, VuGen 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 208.

Checking Scripts Out of the Version Control Database

When you select **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 204.

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 script in, see "Checking Scripts into the Version Control Database" on page 209. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 215.

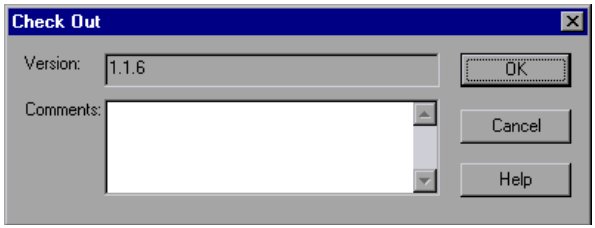
By default, the check out option accesses the latest version of the script. You can also check out older versions of the script. For more information, see "Using the Version History Dialog Box" on page 211.

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 204.

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 Select **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 215.

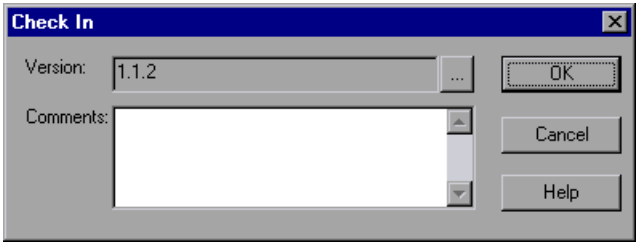
When you check a script back into the version control database, Quality Center deletes the script copy from your checkout directory and unlocks the script in the database so that the script version will be available to other users of the Quality Center project.

To check in the currently open script:

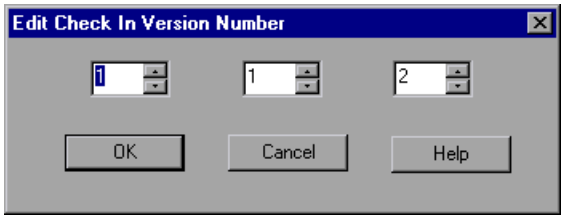
1 Confirm that the currently open script is checked out to you. For more information, see "Viewing Version Information For a Script" on page 211.

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 Select **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.

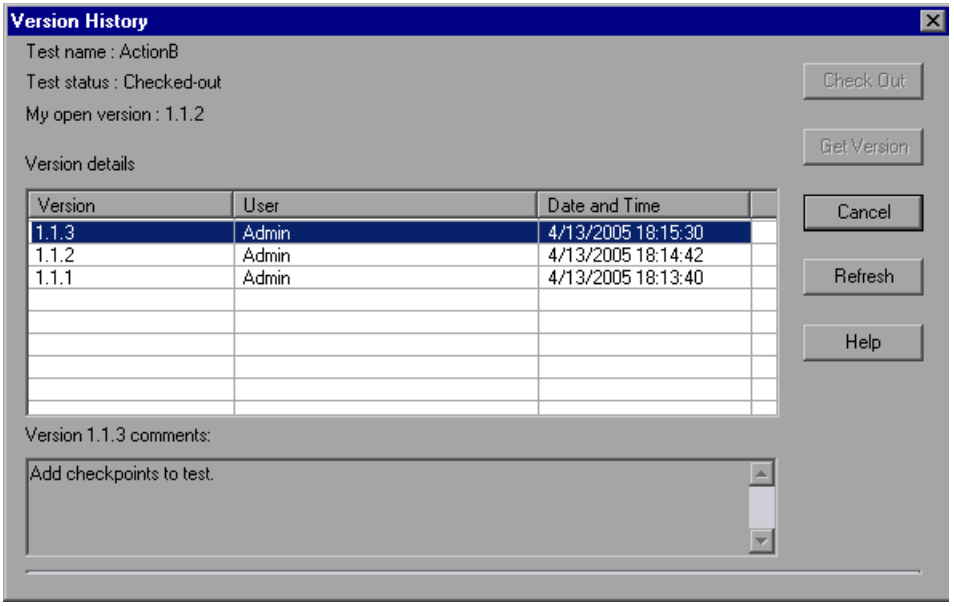
Using the Version History Dialog Box

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 select **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:

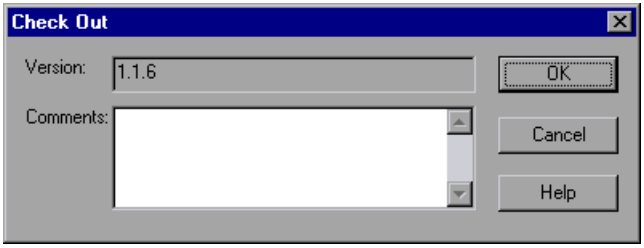
1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 204.
2. Select **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version 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 204.
2. Select **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, select **File** > **Quality Center Version Control** > **Check In**. If you do not want to upload the modified script to Quality Center, select **File** > **Quality Center Version Control** > **Undo Check out**.

   For more information on checking in script, see "Checking Scripts into the Version Control Database" on page 209. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 215.

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. Select **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.

12

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.

This chapter includes:

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, see the *HP Performance Center User Guide*.

Viewing the Vuser Scripts List in Performance Center

In order to run Vuser scripts in Performance Center, they need to appear in the Vuser Scripts list on the Performance Center server. The Vuser Scripts list displays all the scripts that are available for your project.

To open the Vuser Scripts list, select **Vuser Scripts** from the **Project** menu from within the Performance Center User site.

You add scripts to the list by uploading them to the server through the Performance Center interface or directly from VuGen.

For information about uploading scripts through Performance Center, see the *HP Performance Center User Guide*.

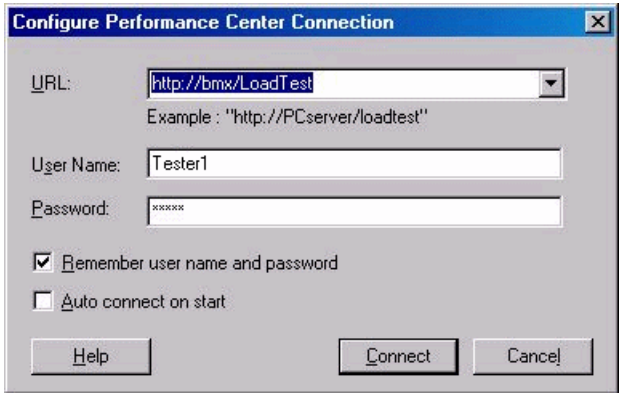
For information about uploading from VuGen, see "Uploading Vuser Scripts" on page 220.

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.

To connect VuGen to Performance Center:

1 In VuGen, select **Tools** > **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 on start**. VuGen attempts to connect to Performance Center using the displayed login information.

6 Click **Connect**. 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.

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.

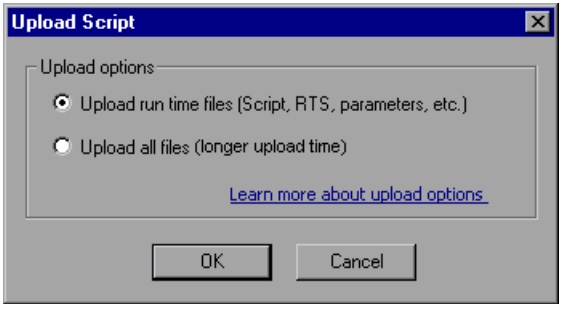
Uploading Vuser Scripts

Transferring a Vuser script from your file system to a server is known as **uploading**. After you upload a script, Performance Center displays it in the Vuser Scripts list.

Before uploading scripts from VuGen to the Performance Center server, you must connect to the server with your credentials. For more information about connecting, see "Connecting VuGen to Performance Center" on page 218.

Upload Options

VuGen lets you select the extent of the uploading. Depending on your requirements, you can do a partial or a complete upload. The upload options are:



- **Upload run time files.** 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.

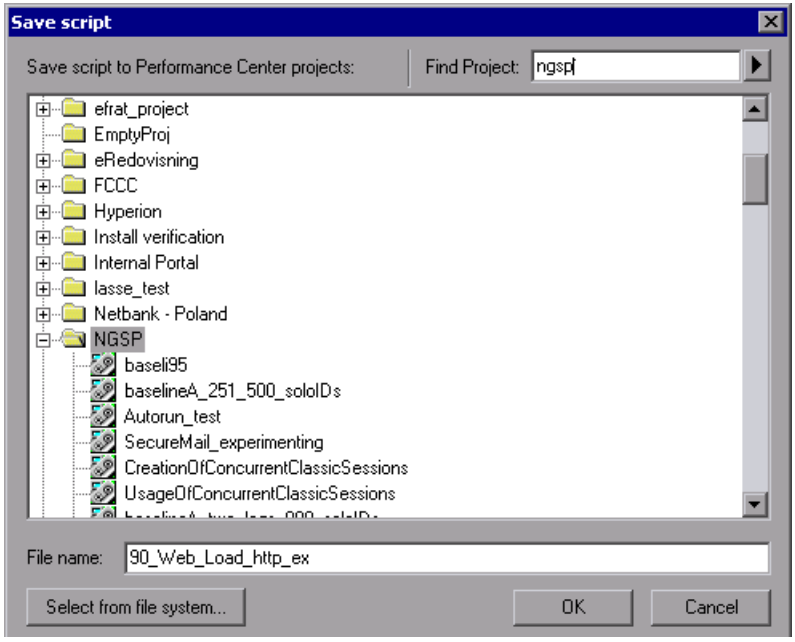
Uploading the run time files only is quicker since VuGen only uploads the script files—not all of the recording data and the replay results.

Note: If you previously downloaded the run time files, by default VuGen will only upload the files that were downloaded. If you want to upload newly created files, for example, snapshots generated after replay, you must specify **Upload all files**.

To upload a script to Performance Center from VuGen:

1 Make sure that you are connected to Performance Center. Select **Tools** > **Performance Center Connection**.

2 Select **File** > **Save As** in VuGen. The Save script dialog box opens.



3 In the **Find Project** box, type a name of a project or part of the name to locate the desired project. To begin the search, click the arrow to the right of the box. To move to the next match, click the arrow again.

4 Select the project under which you want to save the script. In the **File name** box, type a name for the script.

Note: File names can only consist of English letters, digits, or the underscore character, and cannot exceed 250 characters.

5 Click **OK**. The Upload Script dialog box opens. Select one of the Upload Options, **Upload run time files** or **Upload all files**.

6 Click **OK** to upload the files to the Performance Center server.

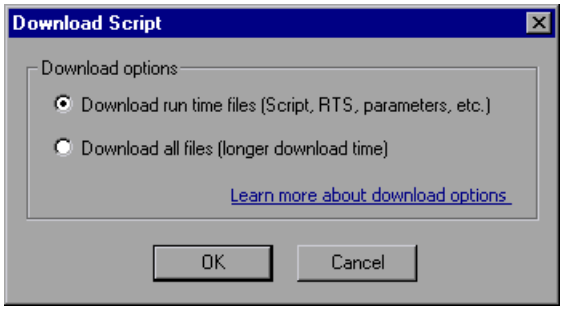
Downloading Vuser Scripts

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 specify whether 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 218.

Download Options

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.

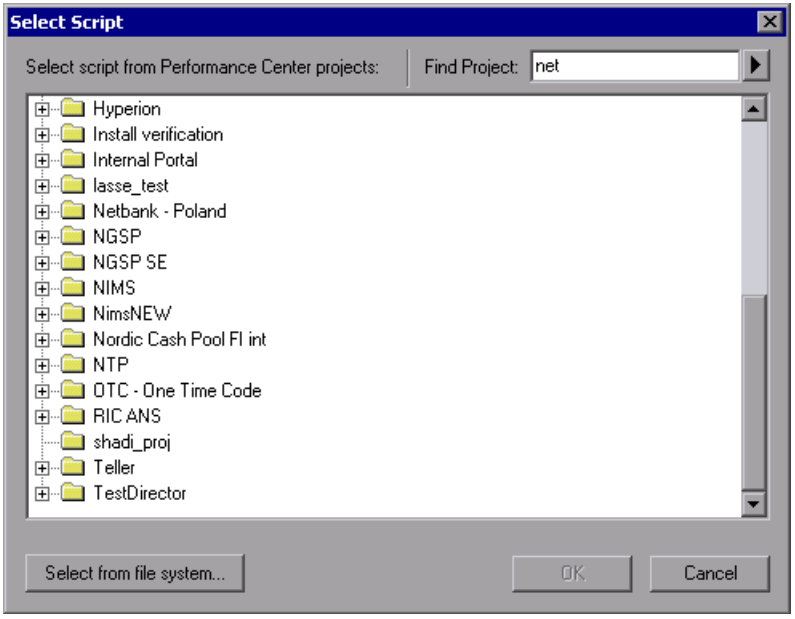
A partial download of run time files only is quicker since VuGen only downloads the script files. If you download all the script and data files, the transfer will take more time.

Once you are connected to the Performance Center server, you can download your script files to VuGen.

To download a Vuser script from Performance Center:

1 Make sure that you are connected to Performance Center. Look for the **PC Connected** button on VuGen's the status bar.

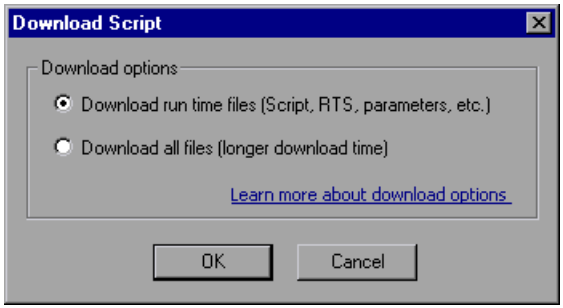
2 In VuGen, select **File** > **Open**. The Select Script dialog box opens.



3 Search for the project containing the script you want to download. In the **Find Project** box, type a name of a project or part of the name to locate the desired project. To begin the search, click the arrow to the right of the box. To move to the next match, click the arrow again.

4 Expand the project and select the script that you want to download.

5 Click **OK**. The Download Script dialog box opens.



6 Select a download option: **Download run time files** or **Download all files**.

7 Click **OK** to download the files from Performance Center. The Downloading files progress bar opens. When the download is complete, the progress bar closes and VuGen displays the script.

Working with Downloaded Scripts

After you download a script, you can edit it in VuGen.

By default, downloaded files are saved to your **Temp** directory. You can save the changes locally or upload the edited script back to the Performance Center server.

If you are connected to Performance Center, the Save operation automatically uploads the script to the Performance Center server.

Downloaded scripts whose source is the Performance Center server, appear in the File menu's Recent Script list, with a **[PC]** prefix.

To save changes made to a downloaded scripts:

1 To save the script on the Performance Center server, click **Save.** VuGen prompts you with the Upload options (provided that you are connected).

If you want to upload newly created files, such as snapshots generated after replay, you must specify **Upload all files**.

2 To save the script locally, click **Save As.** The Save Script dialog box opens. Click **Select from file system** and browse for the desired location. Click **OK**.

Part 2

Protocols

13

Web Services Protocol

You use VuGen to create tests for your Web Services. After creating a Web Services test script, you can view or modify it in either Script view or Tree view.

This chapter includes:

About Web Services Scripts

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 VuGen, 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 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 15, "Web Services - Adding Script Content."

For manual scripts, you use VuGen 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, select 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 15, "Web Services - Adding Script Content."

3 Set properties, values, and checkpoints.

Enhance the script by customizing the step properties, inserting argument values, and setting checkpoints. For details, see Chapter 17, "Web Services Call Properties."

4 Parameterize 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 1091.

5 Enhance your script with transactions.

Define a group of actions as a transaction 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 120.

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 "Web Services JMS Run-Time Settings" on page 331, and Chapter 79, "Configuring Run-Time Settings."

7 Verify that the script is functional.

Replay the script in VuGen to verify that it runs correctly.

For details about replaying the script, see Chapter 9, "Running Vuser Scripts in Standalone 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 281.

After you prepare a script, you are ready to use it for your testing. For more information, see "Using Your Script" on page 280.

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.

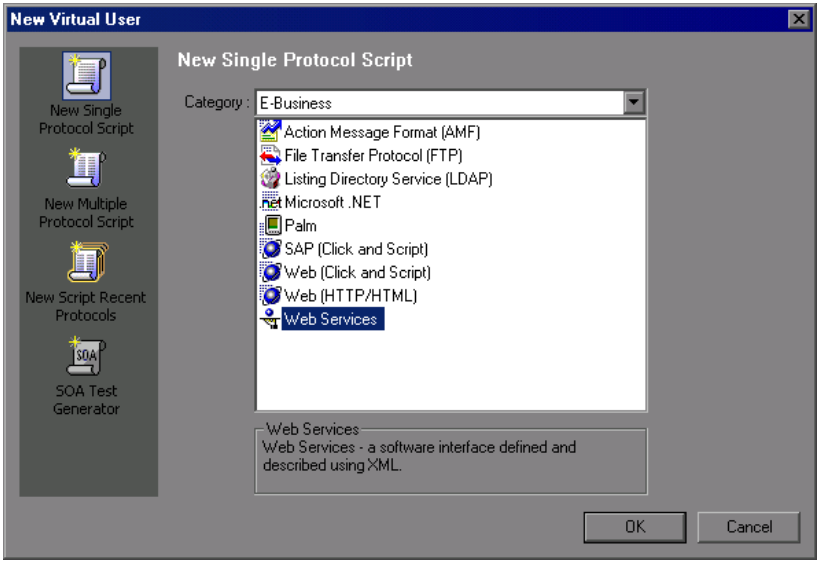
Creating an Empty Web Services Script

To create a script through manual Web Service calls, you must first create an empty script. The empty script serves as a base from which you begin to add Web Service calls and other SOA related steps.

To create an empty script:

1 Create an empty script. Click the New Script button or select **File** > **New** to open the New Virtual User dialog box.

2 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**.

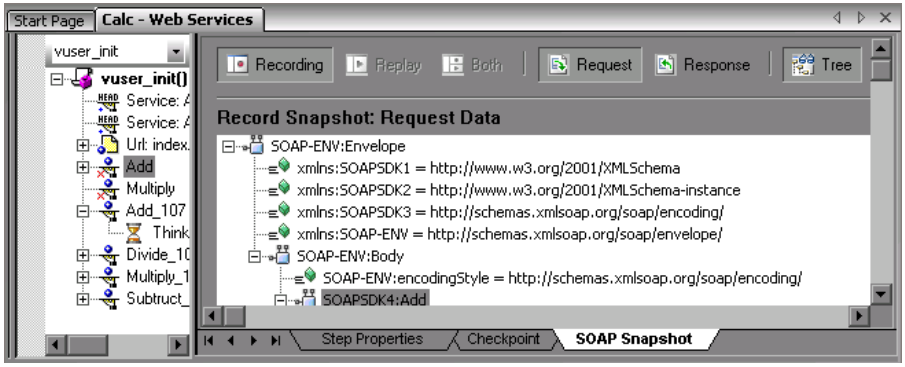
Viewing and Editing Scripts

You can view and edit all of the scripts that you created both manually and automatically in the VuGenwindow.

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, VuGen 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 301.
- **Checkpoint.** A list of checkpoints defined for the step. See "Checkpoints" on page 330.
- **SOAP Snapshot.** A snapshot of the SOAP request and response for both record and replay. See "Viewing Web Services SOAP Snapshots" on page 298.

For more information about these tabs, see Chapter 17, "Web Services Call Properties."

To view a script in Tree view:

1. Click the **Tree** button or select **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 select **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, select **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 275.
7. To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert** > **Add Step** and select the appropriate step. For more information, see Chapter 20, "Web Services - Security."
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 "Checkpoints" on page 330.

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 select **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 select **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 17, "Web Services Call Properties."
4. To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert** > **Add Step** and select the appropriate step. For more information, see Chapter 20, "Web Services - Security."
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, see the *Online Function Reference* (**Help** > **Function Reference**) or select a function and click F1.

Parameterizing Scripts

VuGen 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 70, "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.

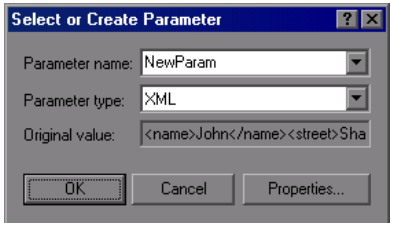
If, however, your arguments are a complex structure with many values, you can use an XML type parameter to replace the entire structure with a single parameter. You can create several value sets for the XML type parameter and assign a different value set for each iteration. For more information, see "XML Parameter Types" on page 1063.

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 334.

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.



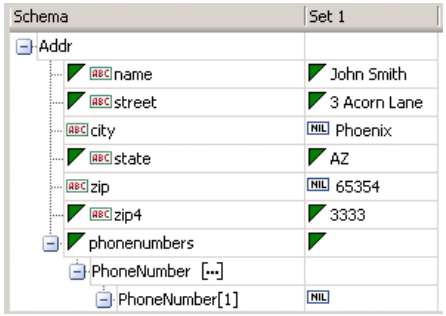
3 Specify a parameter name and type.

4 Click **Properties** to set the type of parameter—File, XML, and so on—and to assign values.

For more information, see Chapter 70, "Working with VuGen Parameters."

XML Editing

Service Test provides an editor that displays the XML structure according to a WSDL or XML schema. The grid-like display lets you view the XML in its proper hierarchy and set values for each of the elements. The left column represents the schema, while the other columns show the XML that is generated and its properties.



While you can use the XML Editor as a standalone editor to format your XML code, it is also integrated with many of the Service Test features, such as parameterization and XML validation.

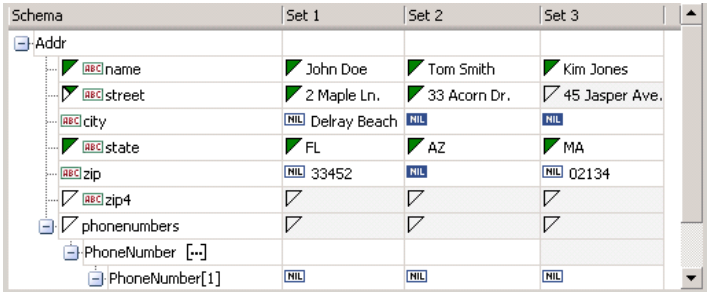
To set values for the elements, you can manually edit the XML or import XML files with the values.

The editor denotes optional elements with a small triangle in the upper left corner of the cell. A filled-in triangle indicates an included element. To exclude an optional element, click the small triangle to clear it.

When you exclude an element, the editor works dynamically and removes the entire element from the XML code. When you re-include the element, the editor adds it back into the XML.

Multiple Value Sets

Value sets are arrays that contain a set of values. You can create multiple value sets for your parameter and use them for different iterations or test runs.



You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

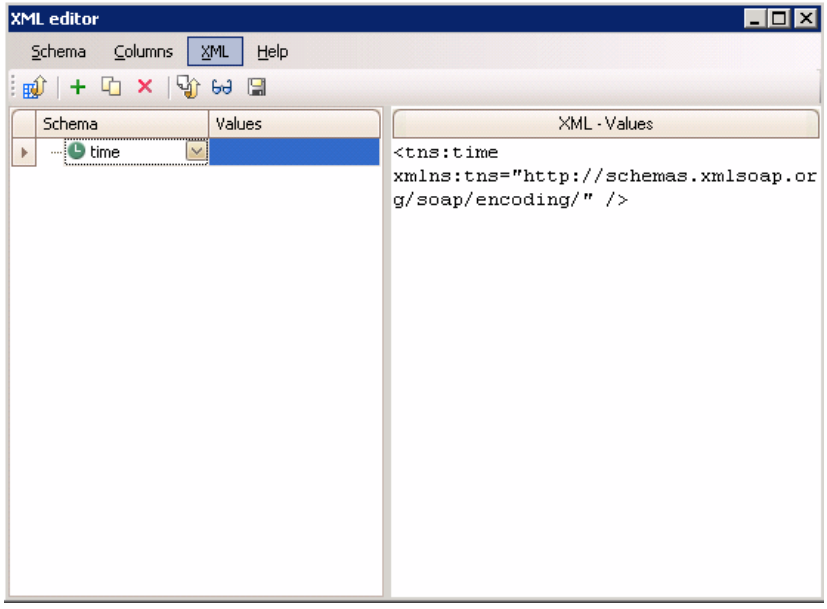
When using value sets, the number of array elements per parameter does not have to be constant. The exception to this is Choice, Derived, and <any> types, where the number of elements is fixed for all value sets.

To resize the columns of the value sets, click the divider in the column's title and drag your mouse to the desired width.

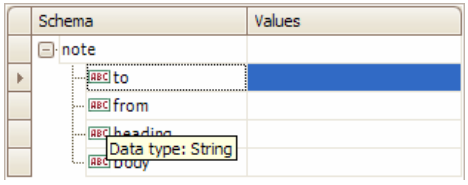
To open the XML editor:

1 Choose **SOA Tools** > **XML Editor**. The editor opens.

2 To load a schema, select **Schema** > **Load**. To load an XML file, choose **XML** > **Load.**

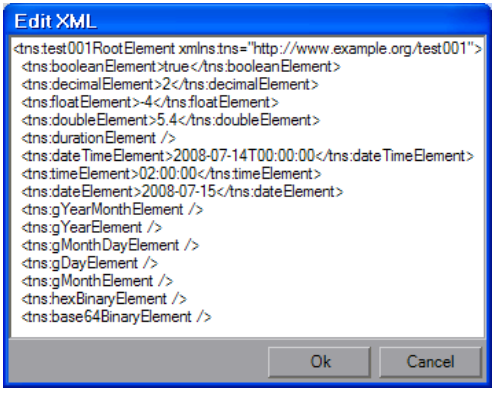


3 Fill in the **Values** column manually, or load values from an XML file. Click the **Load XML from file into the selected column** button. Place your mouse over the element's icon to discover its data type.



4 Choose **Columns** > **Add** or click the Add Column button to add another value set. Add the values manually or by loading XML as described in the previous step.

5 To edit the actual XML code, select a column and click the **Edit XML from the selected column** button. After the editing, click **OK**.



6 To duplicate a column, select it and choose **Columns** > **Duplicate** or click the **Duplicate Column** button.

7 To remove a column, select it and choose **Columns** > **Remove** or click the **Remove Column** button.

8 To save a column to an XML file, select it and choose **XML** > **Save** or click the **Save XML from the selected column** button.

Arrays

You can manipulate array elements and duplicate their whole structure.

VuGen also allows you to include or exclude individual array elements. When you exclude an array element, VuGen excludes it from the SOAP request.

This feature lets you dynamically control the content of the XML by excluding elements in certain value sets, while including them in others. When you exclude an array element, it automatically excludes all of its descendants.

The following example excludes an array element in several value sets.

Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [...]			
PhoneNumber[1]	◆	◆	◆
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	◆	[ ]	◆
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneNumber[3]	◆	◆	[ ]
description	Mobile	Mobile	Mobile
phone-number	555-5555	333-3333	123-4567

To include or exclude array elements, click on the green diamond in the square brackets.

- If the green diamond is visible, it includes the element.
- If the green diamond is removed, it excludes the element and all of its descendants.

Duplicating Arrays

You can duplicate arrays within the grid. VuGen adds an array with an identical structure to the schema column and its value sets.

To duplicate an array, right-click the parent node and select **Duplicate Array Element**.

Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [...]			
PhoneNumber[1]	◆	◆	◆
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	◆	[ ]	◆
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneN...	◆	◆	[ ]
description		bile	Mobile
phone-number		3-3333	123-4567

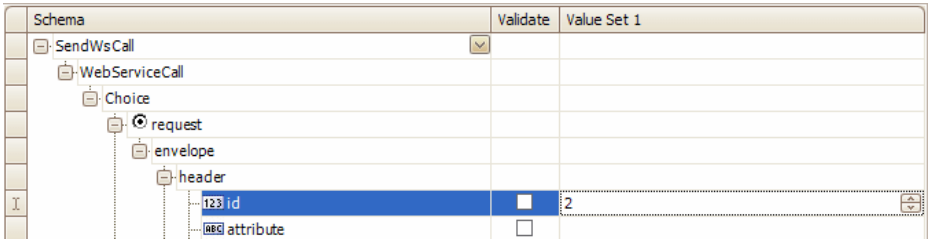
Duplicate Array Element
Remove Array Element

Data Types

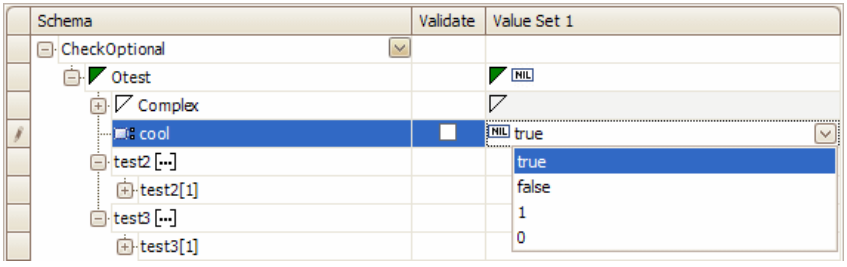
The XML editor lets you manipulate arrays of elements and sub-elements and assign them values.

To determine the actual data type, place your mouse over the icon adjacent to the element name, such as **123**, **ABC**, **B64**, and so forth. The mouseover popup indicates the data type.

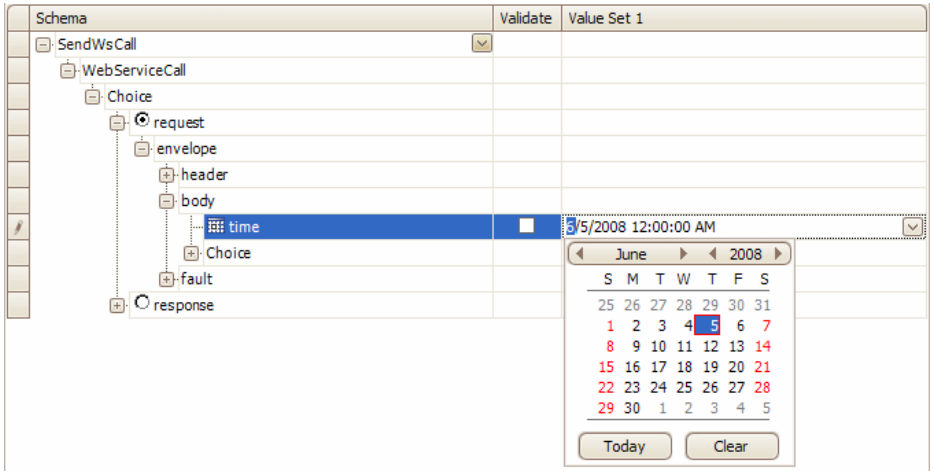
The grid recognizes element data types and provides an interface for setting the values. For example, for an **Int** type, the value cell contains a number scrolling control.



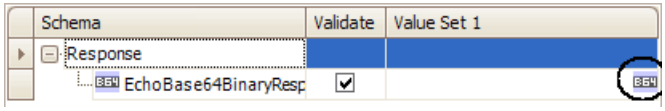
For boolean, it contains a list box with the values **true**, **false**, **1**, or **0**.



For a **Date** element, you can open a calendar to select a date.



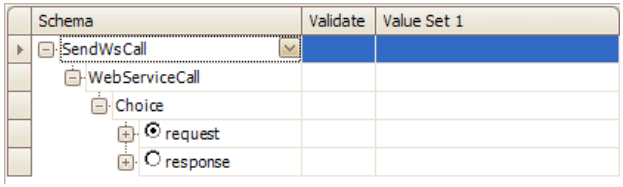
The schema indicates **Base64** elements with a **B64** button to the right of the value box. Click the button to open the Process Base 64 Data dialog box. For more information, see "Base 64 Encoding" on page 316.



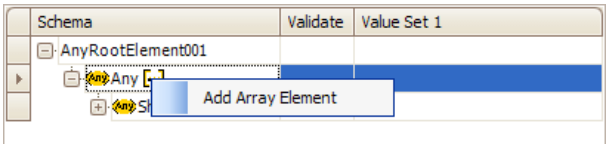
Any Type Elements

Using the XML editor, you can manipulate <**any**> type elements.

For simple, non-array, Any elements, the grid display shows the elements.

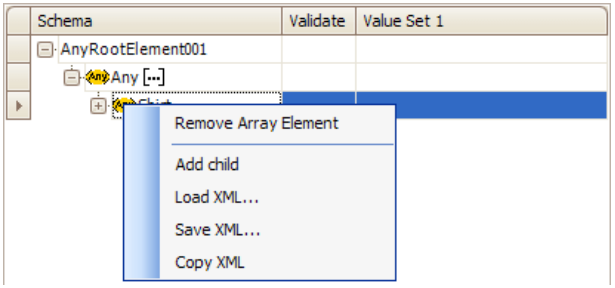


For an array of Any elements, you can manipulate the elements, their names, and their values. Use the right-click menu to add elements.



Manipulating Any Array Elements

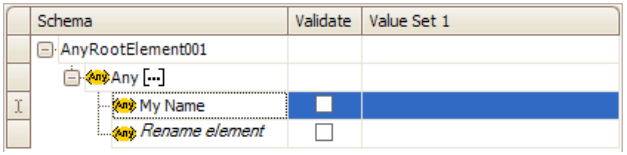
To manipulate Any array elements, open the right-click menu.



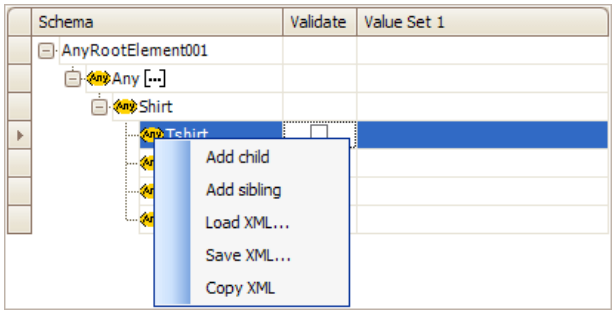
The right click menu provides some or all of the following options, depending on the location of the cursor:

- **Remove Array Element.** Removes the select array element.
- **Add child.** Adds a sub-element to the selected element.
- **Load XML.** Loads the element values from an XML file.
- **Save XML.** Saves the array as an XML file.
- **Copy XML**. Copies the full XML of the selected element to the clipboard.

To provide a name for the Any element, click the *Rename Element* text, and type in a name.

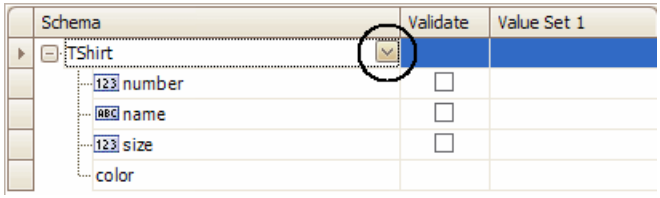


When manipulating a child element, you can use the right-click menu to add a sibling element.



Multiple Roots

If the schema for your Web Service has multiple root elements, you can select one of the elements. Click the small button adjacent to the root name to open the root drop down box.



XML Editor Integration

Besides serving as a standalone editor, the XML Editor is integrated into several of the Service Test components: Parameterization, XML Validation, and Service Emulation.

Parameterization uses the editor to display parameter elements and values as described in "Setting Properties for XML Parameters" on page 1091.

14

Web Services - Managing

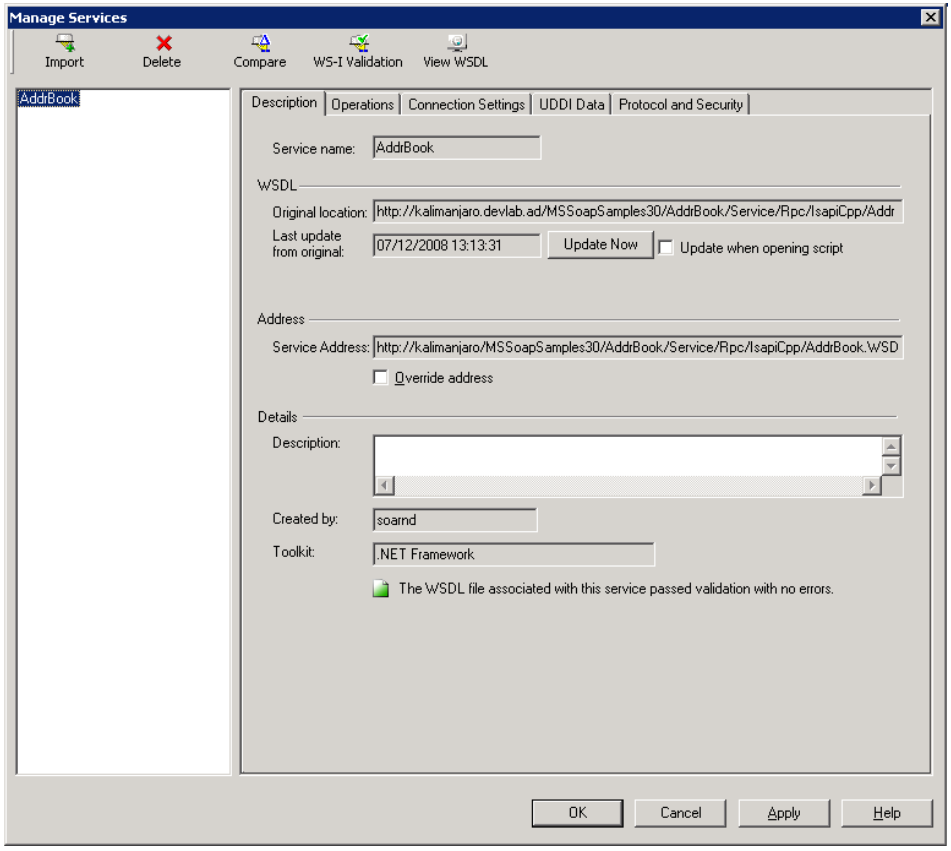
VuGen provides utilities that let you validate and manage the WSDL files associated with your service entries.

This chapter includes:

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, VuGen creates a working copy that it saves with the script—it is not global. Therefore, for each script that you create, you must import the desired WSDL files.

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 249.

To open the Service Management window, select **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
- 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
- Operations
- Connection Settings
- UDDI Data

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**. VuGen backs up the existing WSDL and updates it from the location indicated above.
  - To instruct VuGen 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.

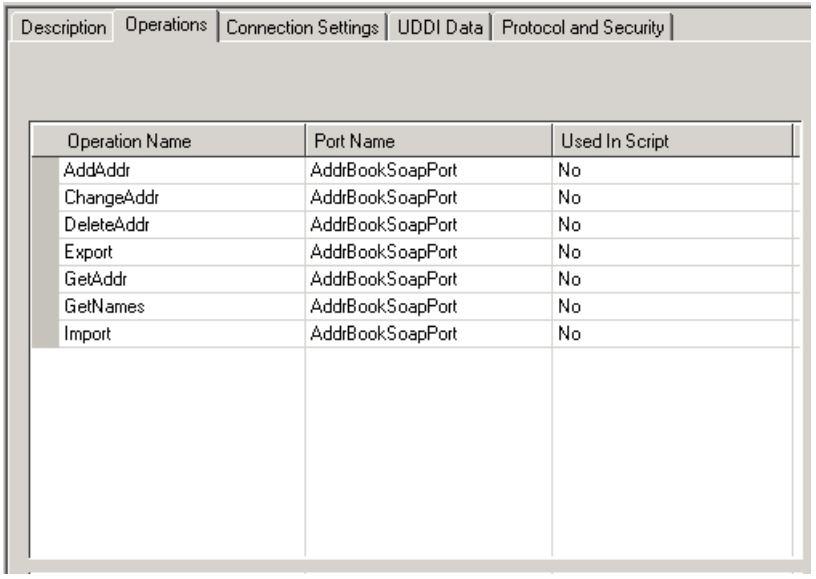
  If you want to override the endpoint specified in the WSDL file, select **Override address** and specify a different address in the **Service address** box.

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.

Operations

Each of the imported services may define multiple operations. The **Operations** tab indicates which operations are being used for the service you selected in the left pane.



You can sort the list of operations by clicking on the relevant column. For example, to list the operations by name, click the **Operation Name** column. To list them in descending order, click the column name again. A small arrow indicates the sorted column. An upward arrow indicates ascending order, while a downward arrow indicates descending order.

Connection Settings

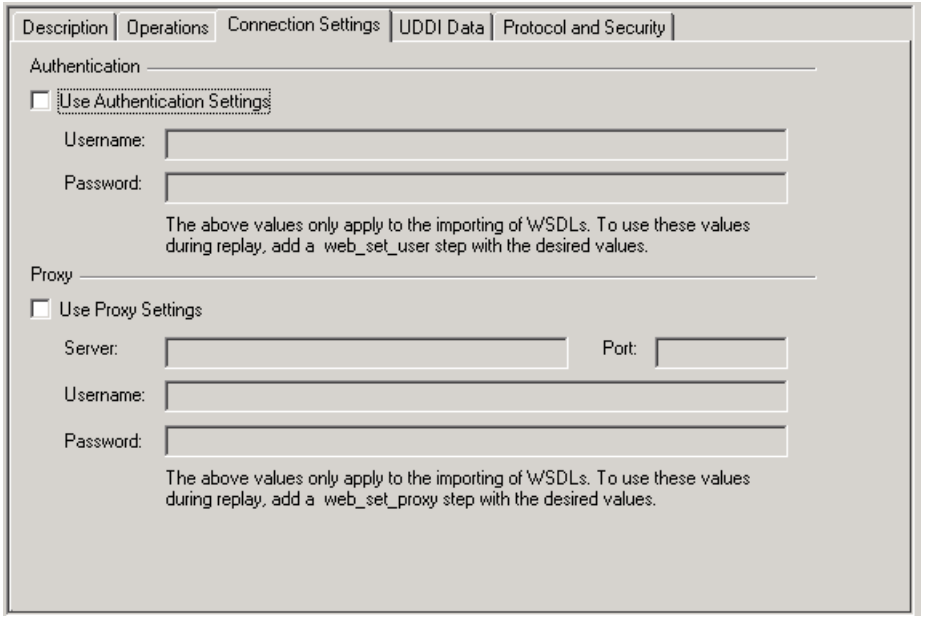
In some cases WSDLs reside on secure sites requiring authentication. In certain instances, the WSDL is accessed through a proxy server.

VuGen supports the importing of WSDLs using security and WSDLs accessed through proxy servers. The following security and authentication methods are supported:

➤ SSL

- Basic and NTLM authentication
- Kerberos for the .NET and Generic toolkits

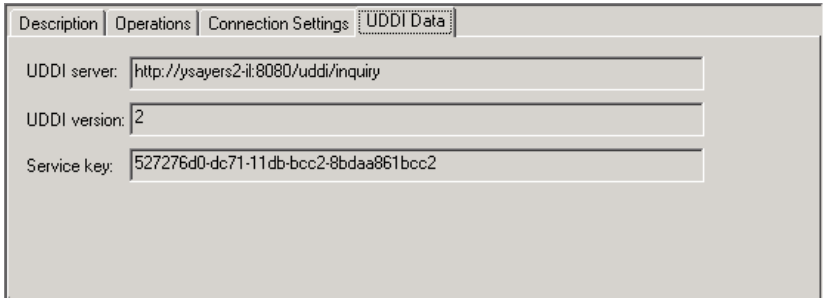
We recommend that you enter the authentication or proxy information while importing the WSDL. If however, the settings changed, you can modify them through the Service Manager's **Connection Settings** tab.



For more information about setting the connection information while importing the WSDL, see "Connection Settings" on page 256.

UDDI Data

You can view the details of the UDDI server for each service that you imported from a UDDI registry.



The read-only information indicates the URL of the UDDI server, the UDDI version, and the Service key.

For information about importing from a UDDI, see "Specifying a Service on a UDDI Server" on page 256.

Protocol and Security Settings

The Protocol and Security Settings tab shows the details of the security scenario applied to the script. If you did not choose a scenario, it uses the default **<no scenario>**.

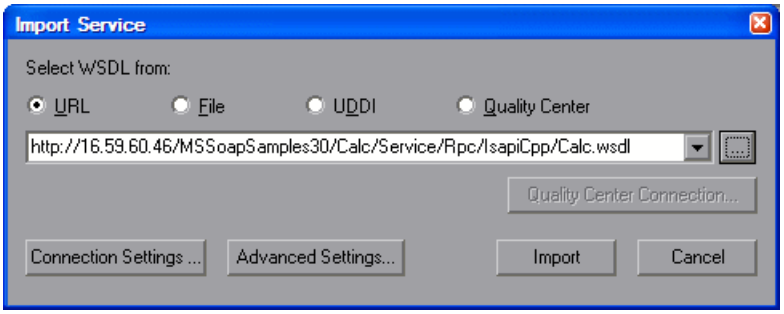
While this tab shows the information as Read-Only, click the **Edit Data** button to modify the security settings. For more information, see "Setting the Security Scenario" on page 374

Importing Services

VuGen lets you import services for the purpose of creating a high-level tests with Web Service Call steps. Typically, you begin creating a script by importing a WSDL file.

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 VuGen detects a problem with your WSDL when attempting to do an import, it issues an alert and prompts you to open the report. The report lists the errors and provides details about them.

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 256.
- **Quality Center.** A service stored in the Quality Center repository. For more information, see "Choosing a Service from Quality Center" on page 257.

VuGensupports 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 256.

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 257.

Toolkit Selection

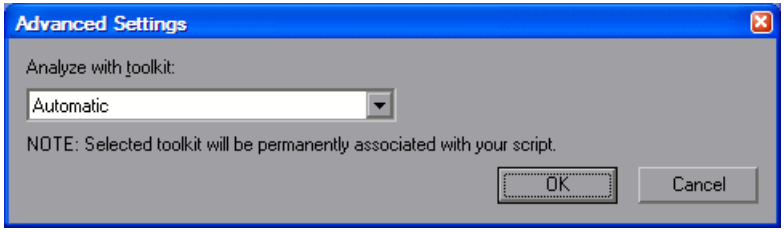
Choosing a toolkit instructs VuGen to send real client traffic using an actual toolkit—not an emulation. Once you select a toolkit, it becomes permanently associated with the script for all subsequent recordings, imports, and replays.

VuGen supports the .NET Framework with WSE 2 version SP3 and Axis/Java based Web Services Framework toolkits. VuGen imports, records, and replays the script using the actual .NET or Axis toolkit.

By default, VuGenuses automatic detection to determine the most appropriate toolkit.

To select a toolkit (for a new script only):

1 In the Import Service dialog box, click **Advanced Settings**.



2 Select a toolkit, or use the default **Automatic** detection.

Connection Settings

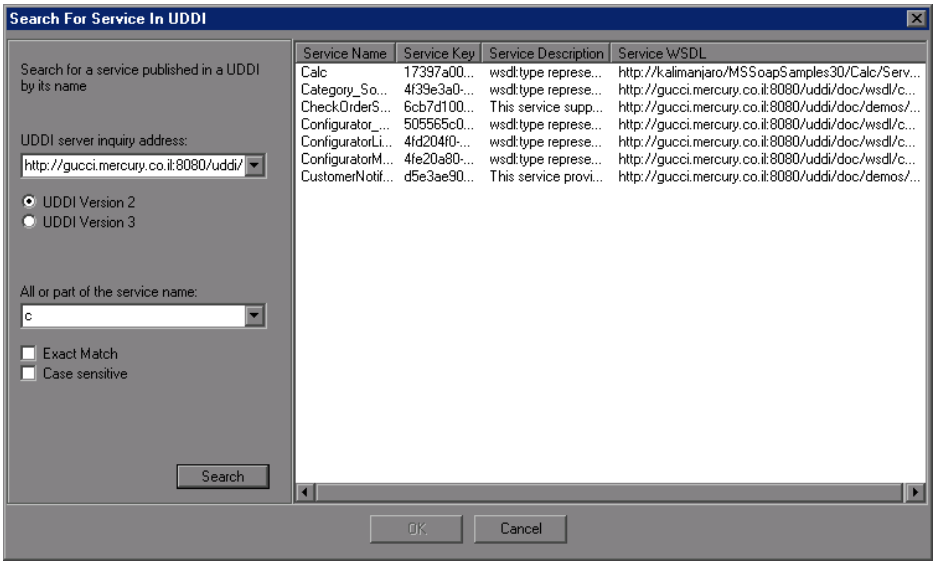
When importing WSDL files from a URL or UDDI, the WSDL may require authentication if it resides in a secure location. In certain cases, the access to the WSDL may be through a proxy server. Using the Connection Settings button, you can specify this information. For more information, see "Specifying WSDL Connection Settings" on page 258.

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**. VuGen 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 VuGen. 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.

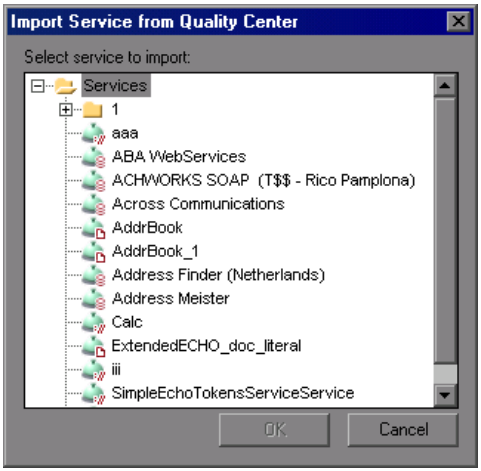
To specify a service from Quality Center:

1 In the Import Service dialog box, select **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 200.

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

VuGen 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.

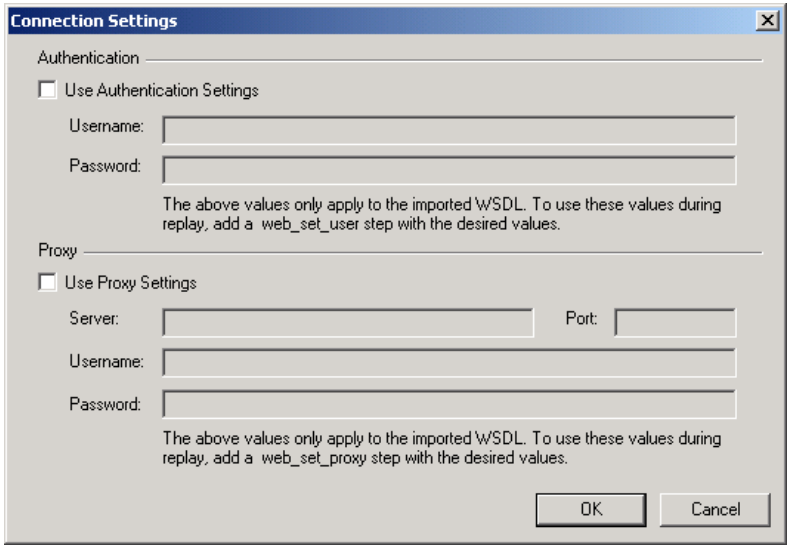
These connection settings only apply to the importing of a WSDL. To use authentication during replay for access to a server, use the **web_set_user** or **web_set_proxy** steps. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

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 to open the box.



4 In the Connections Settings dialog box, select the desired option: **Use Authentication Settings**, **Use Proxy Settings,** or both.

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, VuGen 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, VuGen 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, VuGen 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 inaccurate 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.

VuGen 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.

VuGen also has a general utility that allows you to compare any two XML files. For more information, see "Comparing XML Files" on page 264.

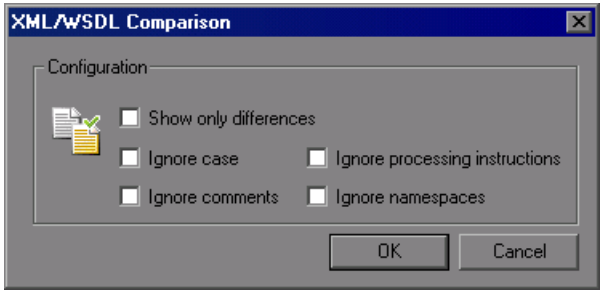
Setting WSDL/XML Comparison Options

VuGen 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. Select **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

VuGen lists the differences between the files in a Comparison report.

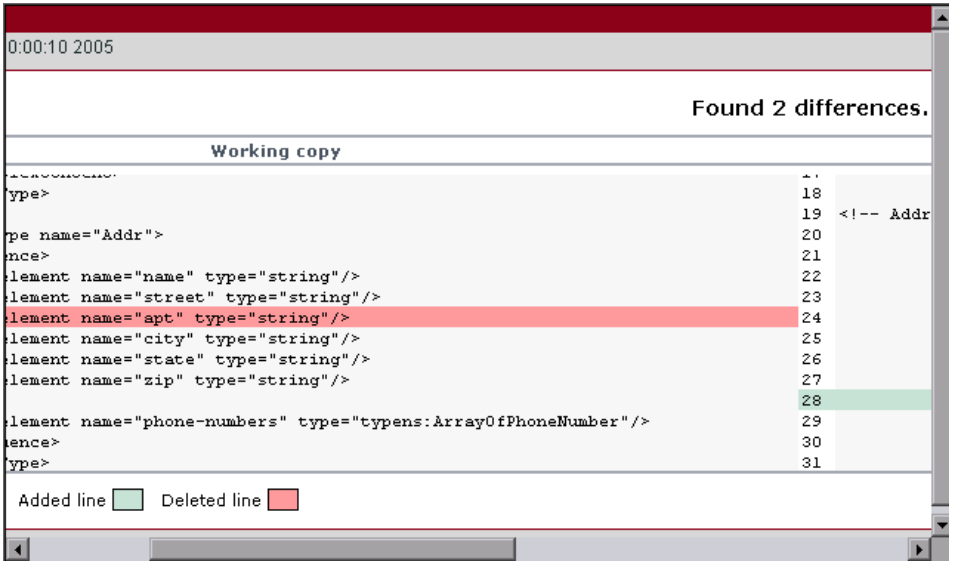
In WSDL Comparison reports, there are two columns— **Working Copy** and **Original File.** The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

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.

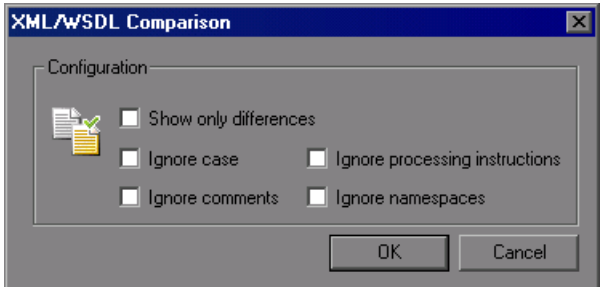


Running a WSDL Comparison

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. Select **SOA Tools** > **SOA Settings** > **XML/WSDL Compare.** The XML/WSDL Comparison dialog box opens. Select the desired options.



2 Open the Service Management window. Select **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 VuGen'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, select **File** > **Exit**.

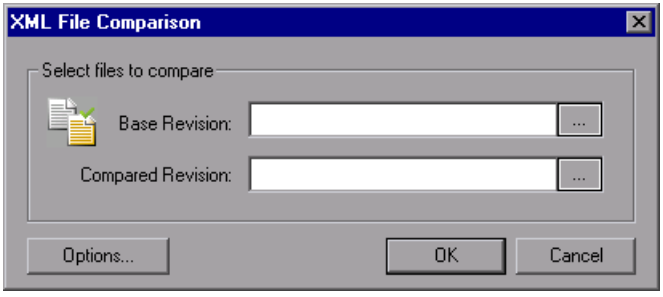
Comparing XML Files

VuGen 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 260.

To compare two XML files:

1 Select **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**. VuGen opens the XML Comparison Report window.

For information about the Comparison report, see "Comparison Reports" on page 262.

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.

15

Web Services - Adding Script Content

You use VuGen to create a script to test your Web Services through recording, manually adding calls, or analyzing server traffic.

This chapter includes:

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 "Creating an Empty Web Services Script" on page 232, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

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 275
importing SOAP requests	"Importing SOAP Requests" on page 277
analyzing server traffic	Chapter 16, "Web Services - Server Traffic Scripts"

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. We recommend that you record with a WSDL when possible.

If you include a WSDL file, VuGen allows you to create a script by selecting the desired methods and entering values for their arguments. VuGen creates a descriptive script that can easily be updated when there are changes in the WSDL.

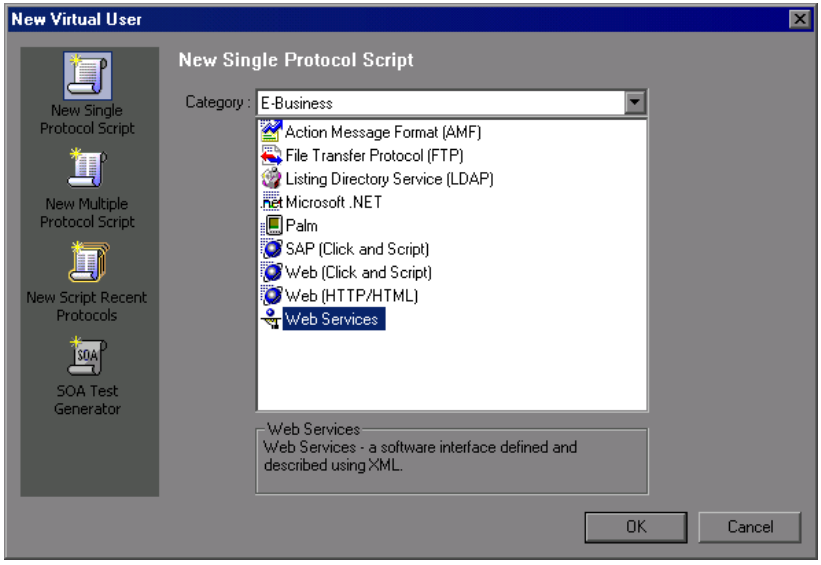
If you do not specify a WSDL file (not recommended), VuGen creates a test with SOAP requests instead of Web Service call steps. If you create a script without importing a service, VuGen creates a **soap_request** step whose arguments are difficult to maintain.

To create Web Services script through recording:

1 Create an empty script.

Select **File** > **New** to open the New Virtual User dialog box.

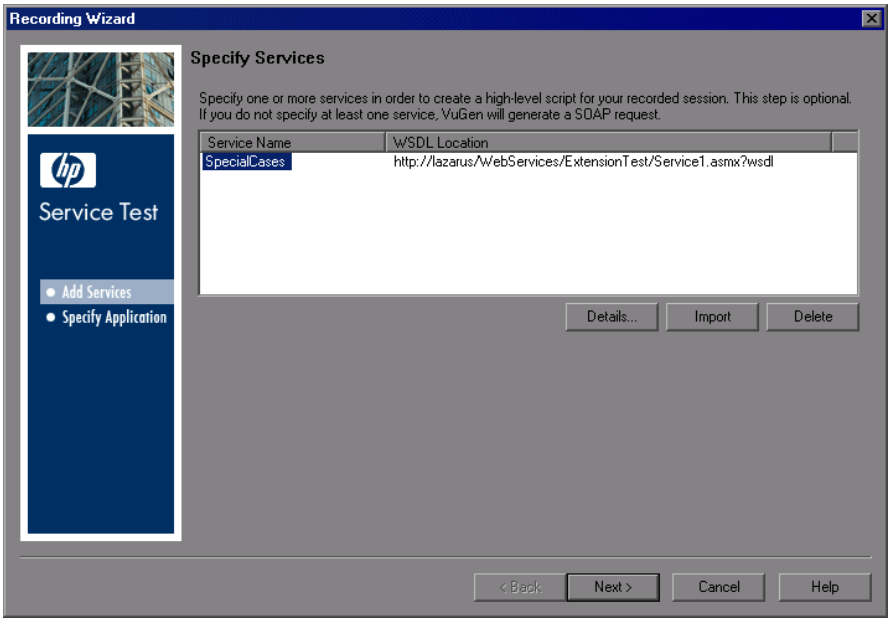
For a single protocol script, click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the E-Business category. Click **OK**.



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**.

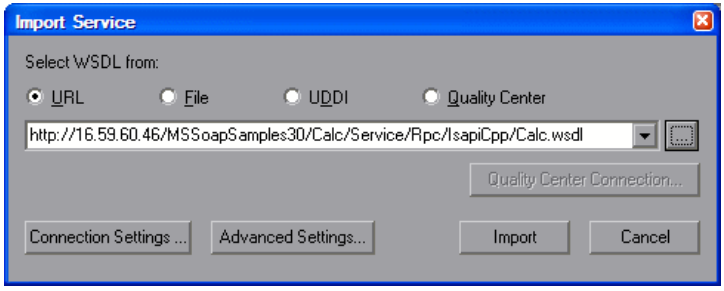
2 Begin the recording process.

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, we recommend to import it here. If you create a script without importing a service, VuGen creates a **soap_request** step. The Import Service dialog box opens.



a Select a source and location for the WSDL.

b Select a toolkit. The toolkit you select is permanently associated with the script. For more information, see "Importing Services" on page 253.

c Click **Import**.

Repeat the above step for each WSDL you want to import.

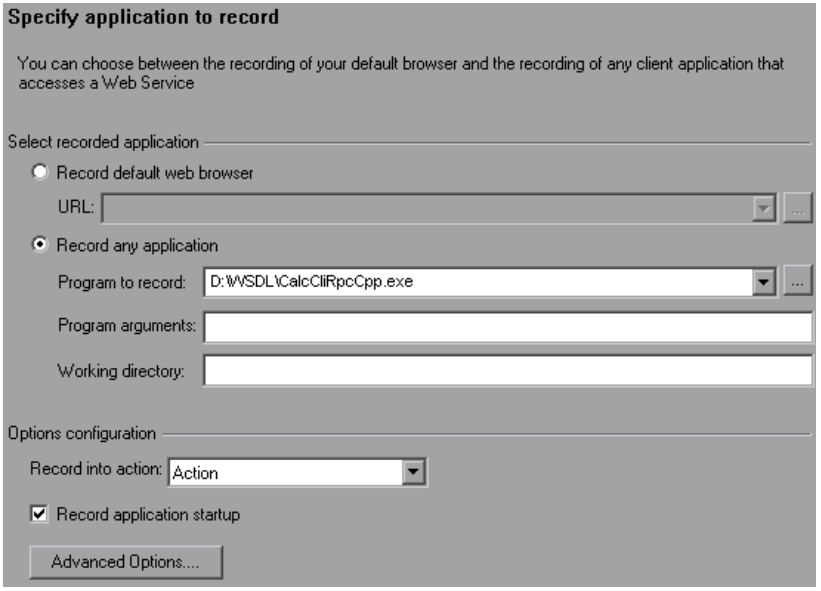
4 Enter details for the Web Service.

Click **Details** to open the Service Manager window and view the details of the Web Services that you added. For more information, see Chapter 17, "Web Services Call Properties."

In the next step, you select 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 the section on setting Script Generation preferences.

3 Click Finish.

VuGen opens the application and begins recording. Perform the desired actions within your application and then press the **Stop** button on the floating toolbar. VuGen 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. You run the completed script in VuGen to check its functionality.You run the completed script in VuGen to check its functionality.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 230.

Viewing the Workflow

When adding script content through recording or Analyzing traffic, VuGen'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 4, "Viewing the VuGen Workflow."

Adding New Web Service Calls

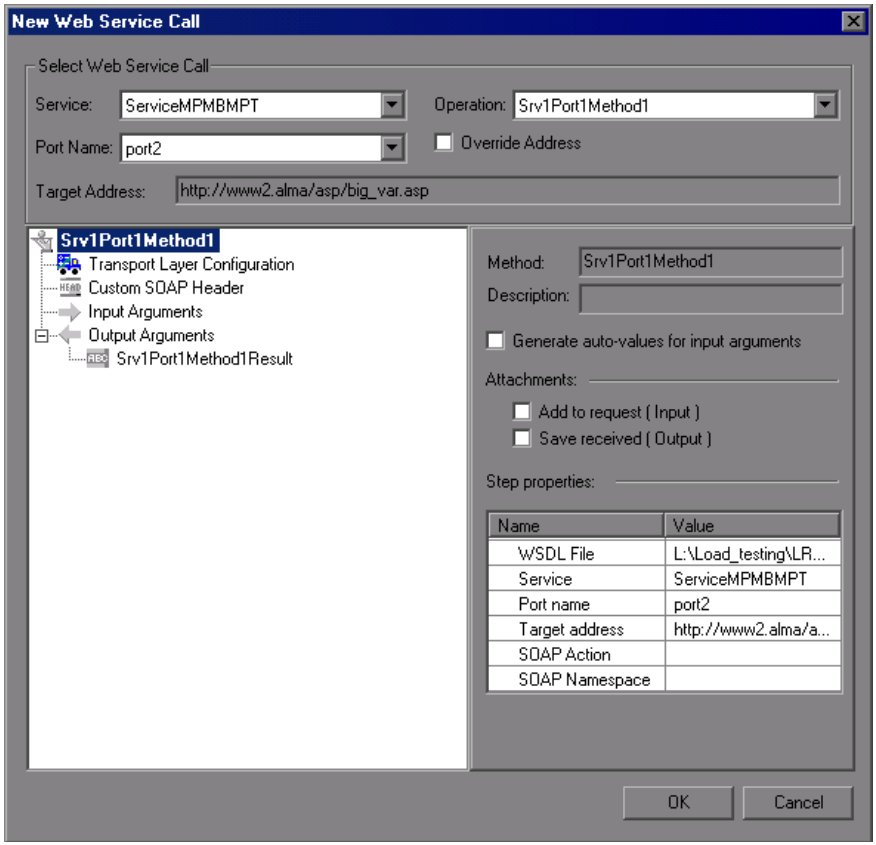
You can add new Web Service call functions in both Tree and Script views.

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 253.

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**. VuGen adds sample values and places an arrow before each of the arguments, to indicate that it is using auto-values.

   To provide sample values for all Input arguments, select the **Input Arguments** node and click **Generate**.

6. To parameterize the input arguments of the operation, see "Setting Properties for XML Parameters" on page 1091.

7. Select the **Transport Layer Configuration** node to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see Chapter 22, "Web Services - Transport Layers and Customizations.".

8. Click on each of the nodes and specify your preferences for argument values. For more information, see Chapter 17, "Web Services Call Properties."

9. 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 320.

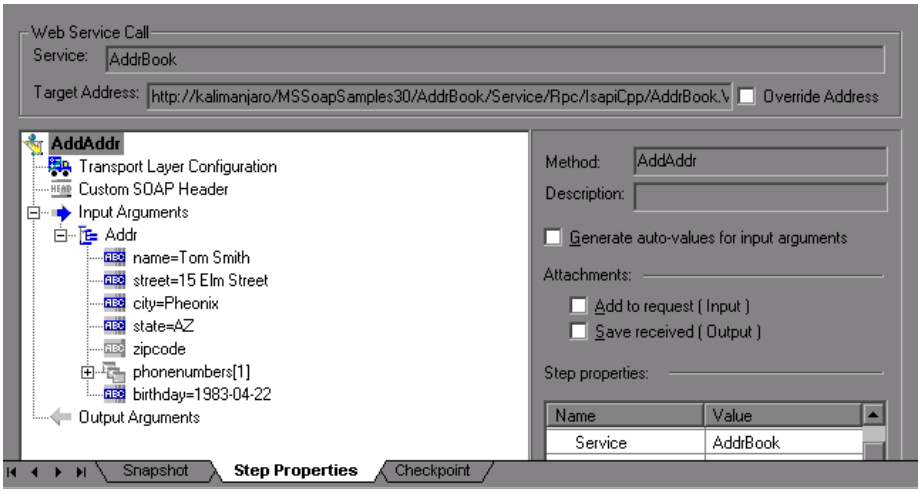
Importing SOAP Requests

VuGen lets you create Web service calls from SOAP files. If you have a SOAP request file, you can load it directly into your script. VuGen imports the entire SOAP request (excluding the security headers) with the argument values as they were defined in the XML elements. By importing the SOAP, you do not need to set argument values manually as in standard Web Service calls.

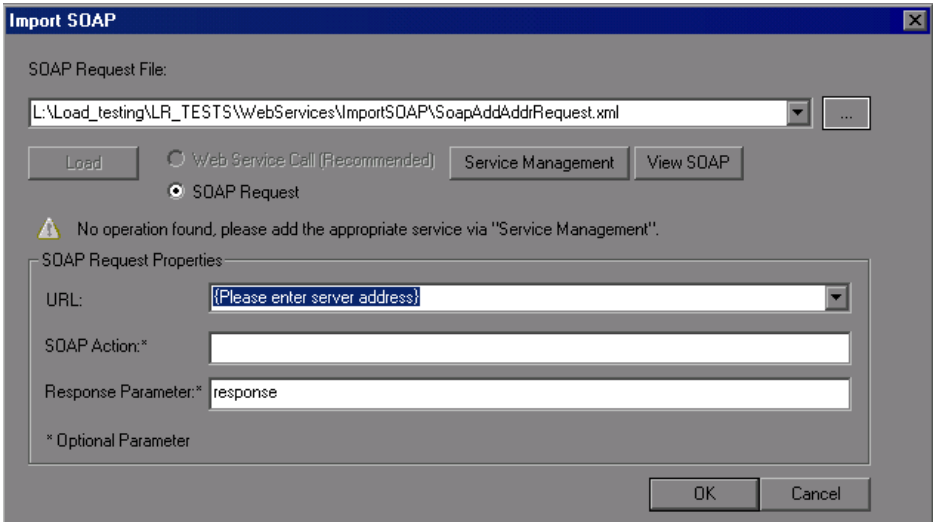
For example, suppose you have a SOAP request with the following elements:

```
- <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  - <q1:AddAddr xmlns:q1="http://tempuri.org/AddrBook/message/">
              <Addr href="#id1" />
    </q1:AddAddr>
   - <q2:Addr id="id1" xsi:type="q2:Addr" xmlns:q2="http://tempuri.org/AddrBook/type/">
          <name xsi:type="xsd:string">Tom Smith</name>
               <street xsi:type="xsd:string">15 Elm Street</street>
               <city xsi:type="xsd:string">Pheonix</city>
              <state xsi:type="xsd:string">AZ</state>
              <zip-code xsi:type="xsd:string">97432</zip-code>
              <phone-numbers href="#id2" />
              <birthday xsi:type="xsd:date">1983-04-22</birthday>
    </q2:Addr>
…
```

When you import the SOAP request, VuGen imports all of the values to the Web Service call:



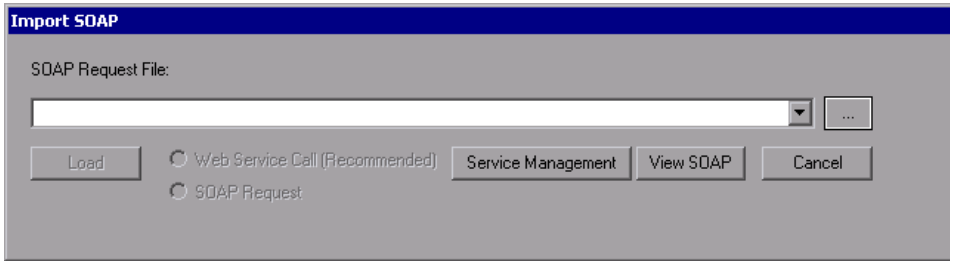
To create a new Web Service call based on a SOAP request, you must first import a WSDL file. If a WSDL is not available, or if you want to send the SOAP traffic directly, you can create a SOAP Request step. You specify the URL of the server, the SOAP action, and the response parameter.



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 select **SOA Tools** > **SOAP Import**.



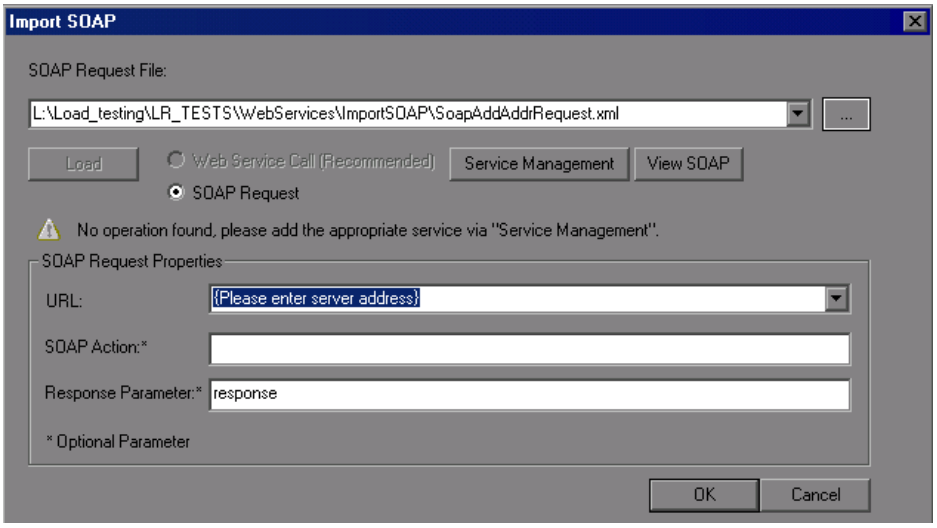
2 Browse for the XML file that represents your SOAP request.

3 Select the type of step you would like to generate: **Create Web Service Call** or **Create SOAP Request**. In order to create a Web Service Call, you must first import at least one WSDL that describes the operation in the SOAP request file. To 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**. VuGen 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 301.

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 select the service whose SOAP traffic you want to import. For information about additional properties, see "Understanding Web Service Call Properties" on page 301.

6 Click **OK** to generate the new step within your script.

7 Set checkpoints and replay the step. For more information, see "Checkpoints" on page 330.

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 281.

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 18, "Web Services - Preparing for Replay."
- **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, see the *HP Business Availability Center* documentation.

Managing Data in Quality Center

When working with Quality Center, you can assign different parameter values for each instance of the Quality Center test set. This enables you to run the same test with different data.

When you modify the parameter values for one test instance, it does not affect other test instances. At any point, you can restore the original parameter values.

For more information, see the *HP Quality Center User 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-in 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, see the *HP Service Test Management User Guide*.

16

Web Services - Server Traffic Scripts

Using VuGen, you can create scripts to test your Web Service by analyzing server traffic capture files.

This chapter includes:

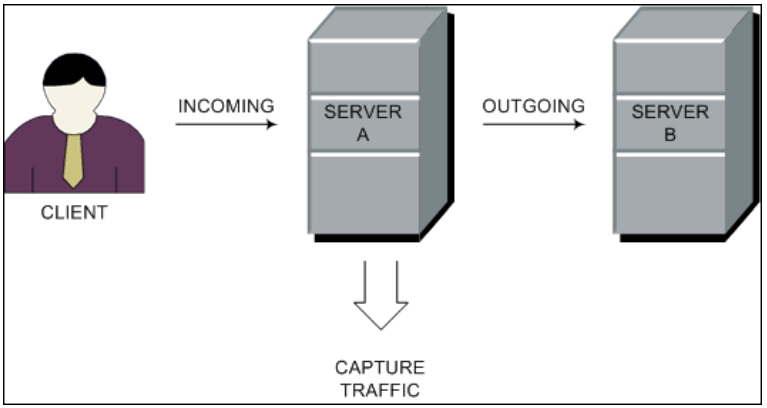
The following information only applies to Web Services/SOA Vuser scripts.

About Creating Server Traffic Scripts

The main focus when testing enterprises and complex systems, is to measure the performance from the client end. Ordinarily, VuGen records the actions you perform in the application or browser, and generates a script emulating the client actions and requests to the server.

In certain test environments, you may be unable to record the client application to retrieve the requests to the server. This may be a result of the server acting as a client, or because you do not have access to the client application. In these cases, you can create a script using VuGen's **Analyze Traffic** feature.

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server. The steps in creating a script by analyzing server traffic are described 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. VuGen examines all of the traffic going into the server, extracts the relevant messages, and creates a script. In the above diagram, if the client is unavailable, you could create an Incoming script to emulate the requests coming into **Server A**.

Outgoing Traffic scripts emulate the server acting as a client for another server. In an application server that contains several internal servers, you may want to emulate communication between server machines, for example between **Server A** and **Server B** in the above diagram. The solution in this case is to generate a script from the traffic sent as output **from** a particular server.

When you create an Outgoing traffic script, you indicate the IP address of the server whose outgoing traffic you want to emulate, and VuGen extracts the traffic going out of that server. In the above diagram, an Outgoing script could emulate the requests that **Server A** submits to the **Server B**.

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.

VuGen uses the capture file to analyze the server traffic and emulate it. For more information, see "Generating a Capture File" on page 288.

2 Create a new Web Services script.

Using VuGen's main interface, you create a new Web Services script. For more information see Chapter 15, "Web Services - Adding Script Content."

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.

Click the **Analyze Traffic** button. 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 291.

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 292.

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 294.

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.

The Capture File Command Line Utility

The VuGen 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. Select **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 VuGen.

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.

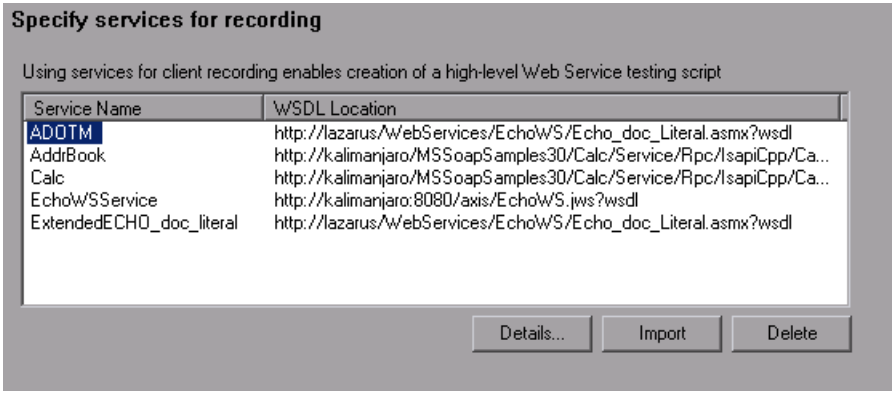
Creating a Basic Script from Server Traffic

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, VuGen will create a script with **web_service_call** functions. If you do not specify a service, VuGen creates a script with **soap_request** functions.

To create a server traffic script:

1. Select **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 select **Vuser** > **Analyze Traffic**. The wizard opens.
4. Add one or more services to the list. This step is optional.



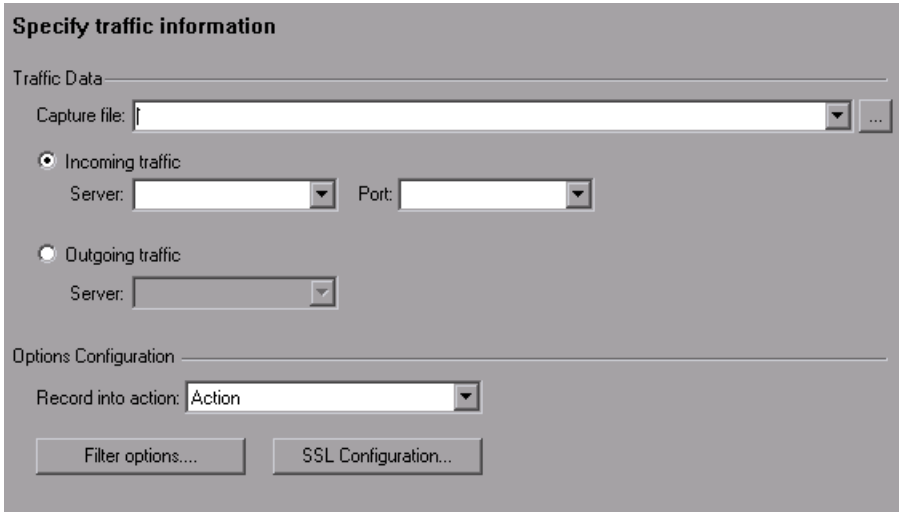
- ➤ To add a new service, click **Import**. In the Import Service dialog box, specify the location of the WSDL. You can specify a URL, File, UDDI server (such as Systinet), or a location in Quality Center. In the Import Service dialog box, you also select a toolkit for analyzing the service. The selected toolkit will be permanently associated with the script—it cannot be changed. For more information, see "Importing Services" on page 253.
- ➤ To set or view details about the services, click **Details** to open the Service Management window. For more information about Service Management, see Chapter 14, "Web Services - Managing."
- ➤ 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.



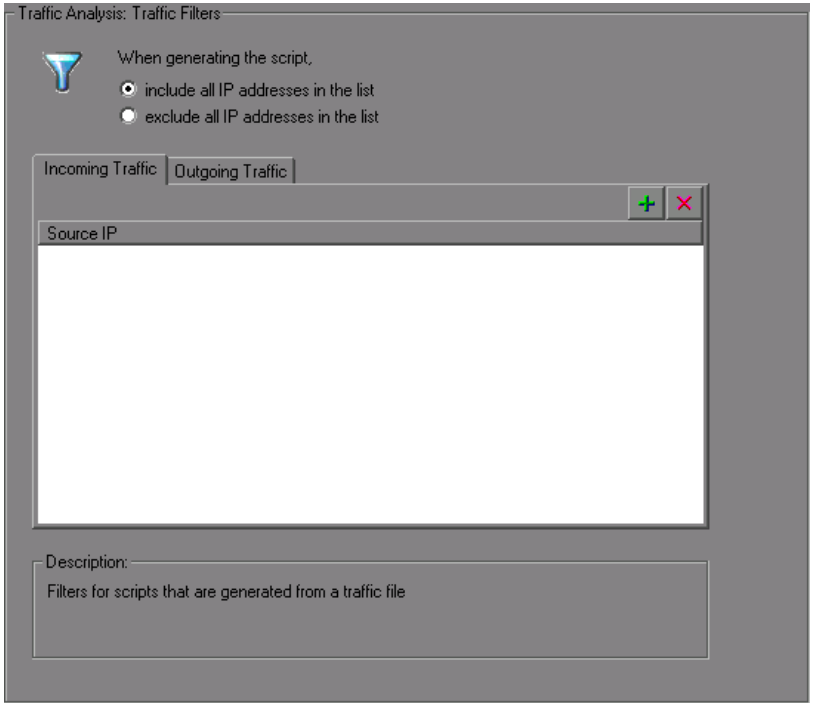
- **Capture file.** The name and path of the traffic file, usually with a cap extension.
- **Incoming traffic:Server/Port.** The IP address and port of the server whose incoming traffic you want to examine.
- **Outgoing traffic:Server.** The IP address of the server whose outgoing traffic you want to 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 294.

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 VuGen. 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 select **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.

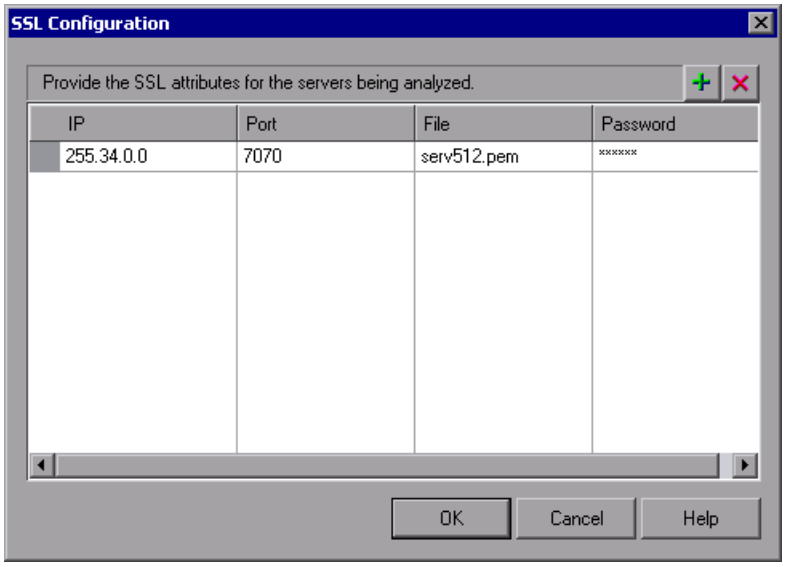
If the traffic is SSL encrypted, you must supply a certificate file and password for decryption. If you want traffic from multiple servers to be reflected in the script, you must supply a separate certificate and password for each IP address that uses SSL.

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.

17

Web Services Call Properties

You use the Web Service Call view to display snapshots, set properties, and add checkpoints to Web Service calls.

This chapter includes:

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. Select **View > Tree View** in the VuGen 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 301.

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 "Checkpoints" on page 330.

Viewing Web Services SOAP Snapshots

You can use VuGen'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:

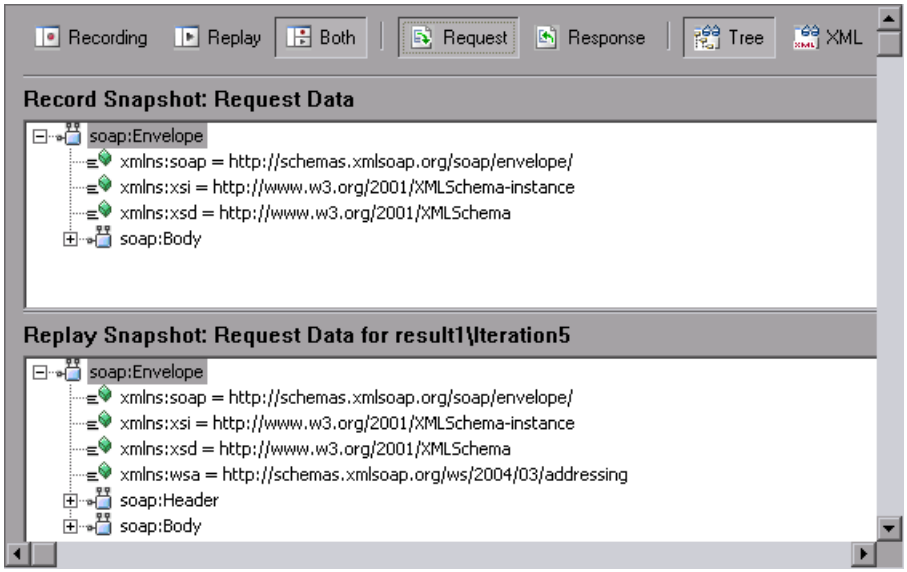
- Record and/or Replay
- Request or Response Data
- Tree or XML View

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.

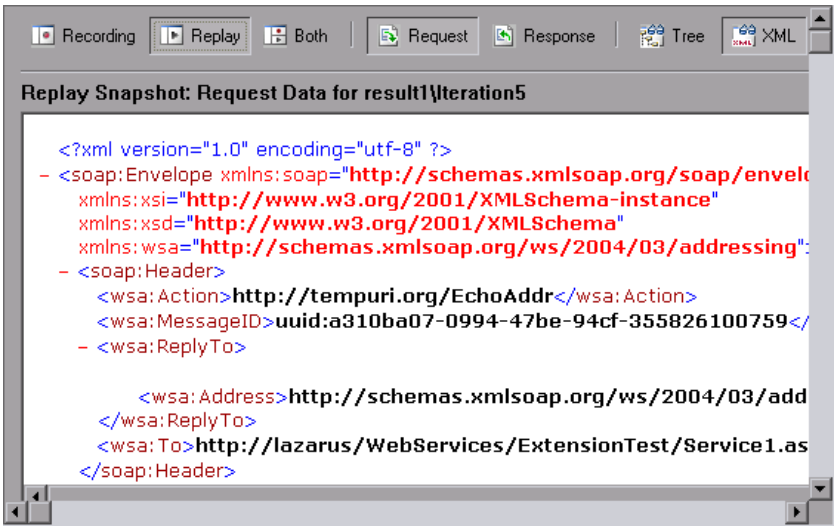
In **Request** view, the displayed values are those that were sent to the server during the Web Services session. In **Response** view, the displayed values are the results returned by the server.

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 select **Node Properties** from the right-click menu.

In the **XML view**, you can view the whole SOAP message in XML format.



Specifying 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 specify an iteration to display:

1. Select **View** > **Snapshot** > **Choose Iteration**.
2. Select the desired iteration and click **OK**.
3. To display results from another folder, select **Select Folder** and browse to the location of the test results.

Understanding Web Service Call Properties

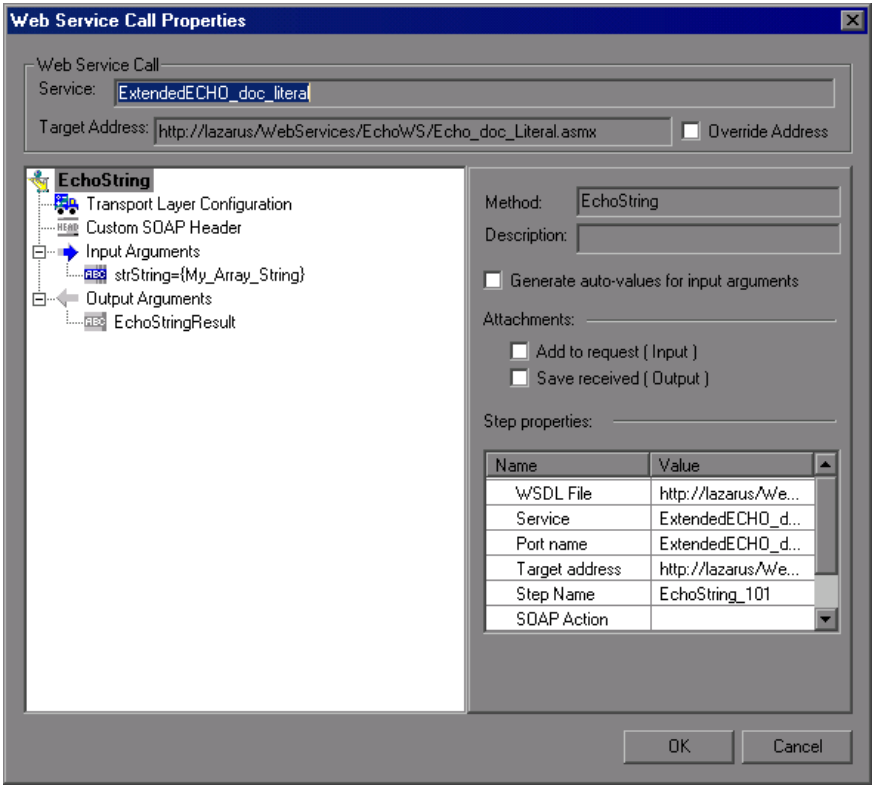
VuGen provides an interface for you to view and modify the properties of each one of the Web Service calls.

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	➤ A checkbox to include input/output attachments ➤ The step properties

Transport Layer Configuration	Advanced transport options: ➤ **HTTP/S Transport** with **Async** or **WSA** support. ➤ **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 325.
Input Arguments node	➤ The **Name** of each method and its **Value.** ➤ **Include All**, **Reset** and **Generate** buttons.
An individual argument	➤ **Name.** The name of the argument (read-only) ➤ **Type.** The argument type as defined in the WSDL. When the WSDL contains derived types, this box becomes a drop down list. For more information, see "Derived Types" on page 311. ➤ **Include argument in call.** Includes the Optional parameters in the Web Service call. See "Working with Optional Parameters" on page 312. ➤ **Nil.** Sets the Nillable attribute to True. ➤ **Value.** The value of the argument. For base64 binary type arguments, **Get from file** or **Base64 Encoded text**. ➤ **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 277. ➤ **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 output argument list ➤ Negative Testing options, as described in Chapter 24, "Web Services - Negative Testing"
Input Attachments	The **Attachment Format** for encoding the soap request: DIME or MIME for the VuGen toolkit, DIME only for .NET (see "Including MIME Attachments" on page 429), 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 VuGen to detect it automatically or select a type from the dropdown list or enter a value manually. ➤ **Content ID.** The attachment's ID attribute. You can instruct VuGen 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

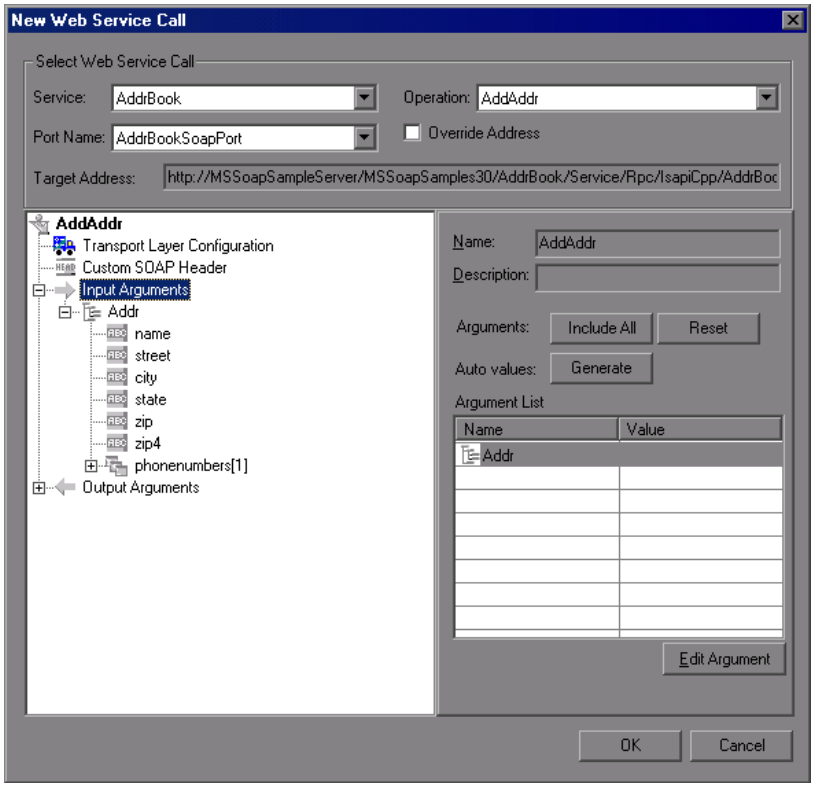
➤ Output Arguments

➤ Arrays

- Attachments
- SOAP Headers

Input Argument Values

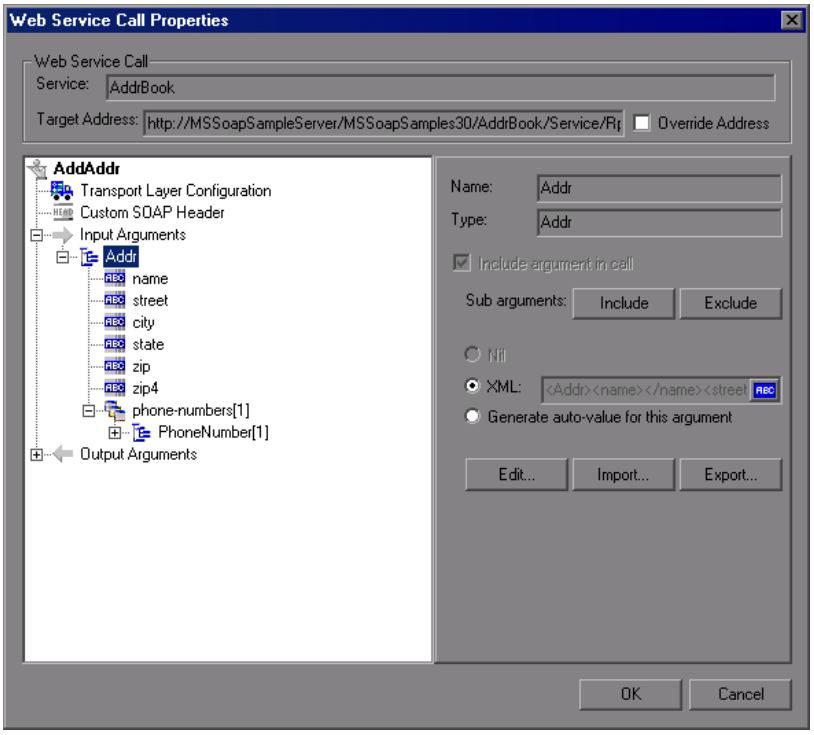
The Input Arguments node lets you define values for all of the input arguments and lets you control the Optional elements. For more information about Optional elements, see "Working with Optional Parameters" on page 312.



- **Include All.** Includes all Optional elements—all of the operation's elements are included.
- **Reset.** Excludes all Optional elements and only includes the mandatory ones.

- **Generate.** Includes all Optional elements and generates automatic values for all of the operation's elements.
- **Edit Argument.** Opens the node of the selected argument and lets you set its values.

The individual argument nodes lets you define values for each of the input arguments.

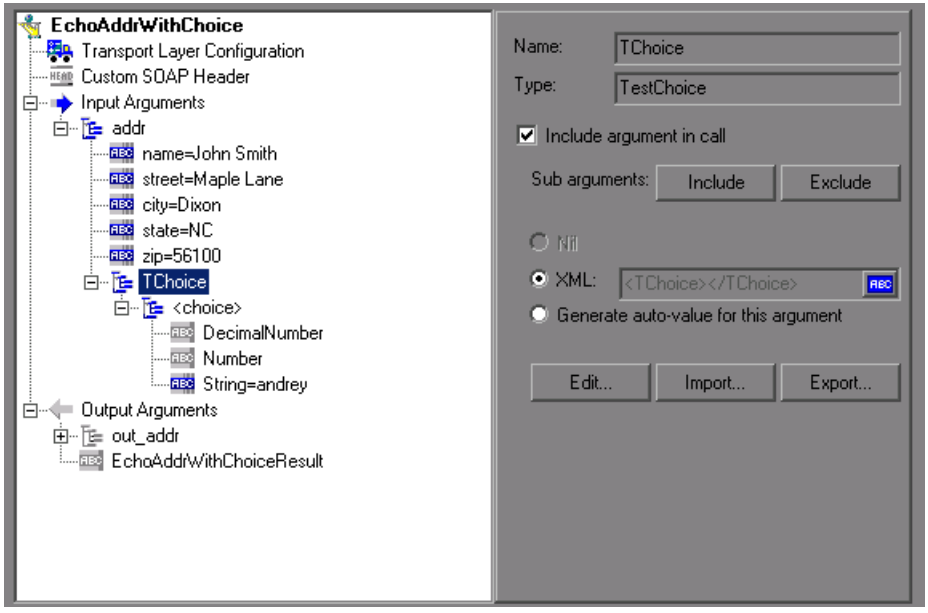


- **XML/Value.** A manually specified value for the node. If your argument is an array, you can specify an XML structure. Otherwise, specify an ordinary value. To create a parameter for the argument, click the **ABC** icon in the right corner of the **XML/Value** box to open the Select or Create Parameter dialog box.

➤ **Generate auto-value for this argument.** If you want VuGen to automatically generate a value for this argument, select this option or select the argument in the tree hierarchy and select **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.

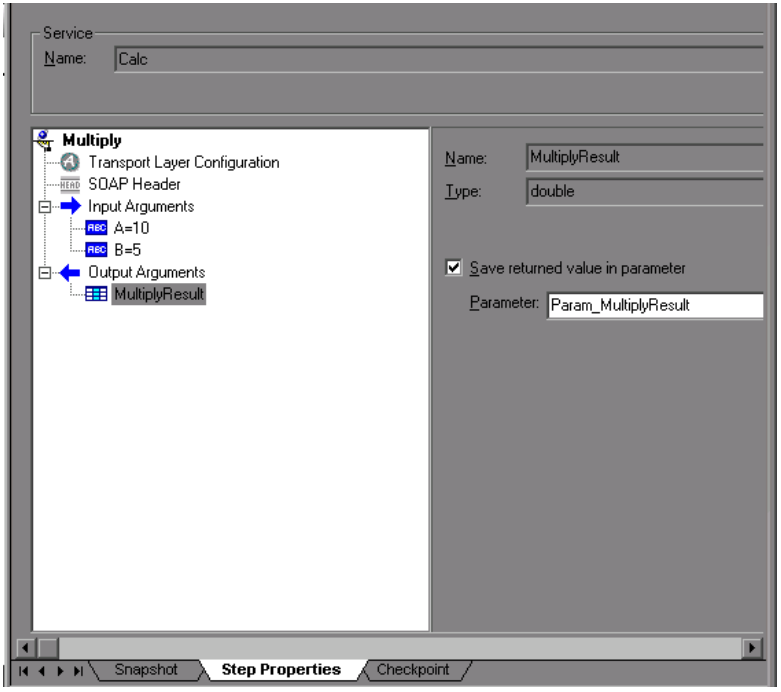
To parameterize the argument, click the **ABC** icon. In the Parameter Properties dialog box, provide values for the choice argument. You only need to provide values for one of the choice elements. When running multiple iterations, the script uses the values for the same choice element, according to the assignment method (sequential, unique or random). For example, if your choice elements are **Decimal Number**, **String**, and **Number**, 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.

For information about working with optional arguments, see "Working with Optional Parameters" on page 312.

Output Arguments

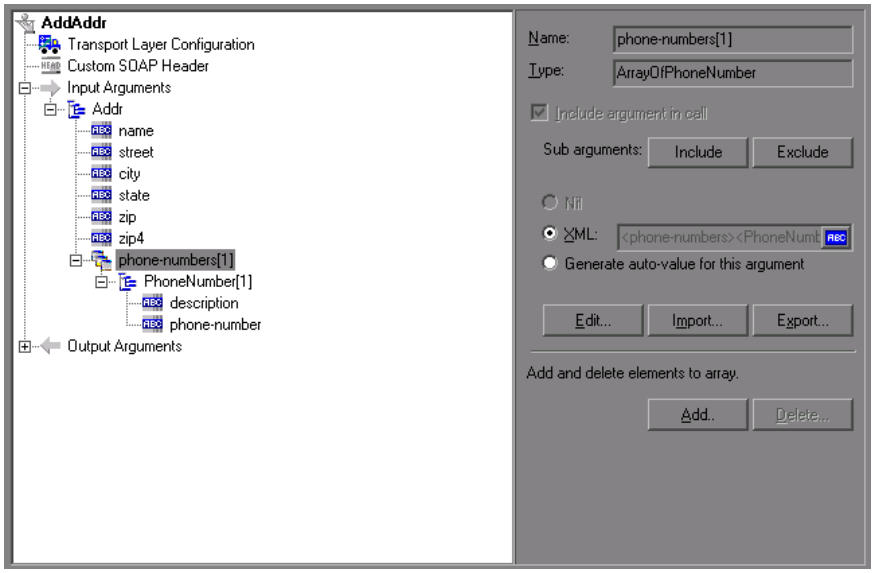
You can view the output argument values and save them to parameters or in an array.



➤ **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.

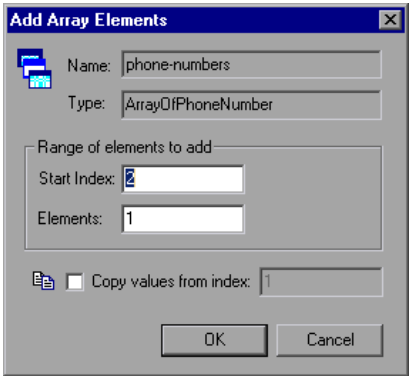
When saving an array to a parameter, the number of array elements per parameter is constant. If you want to run multiple iterations, with each iteration using a different number of array elements, you need to define separate parameters, each containing the desired number of array elements.

For more information about XML parameters, see "Setting Properties for XML Parameters" on page 1091.

- **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 277.
- **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

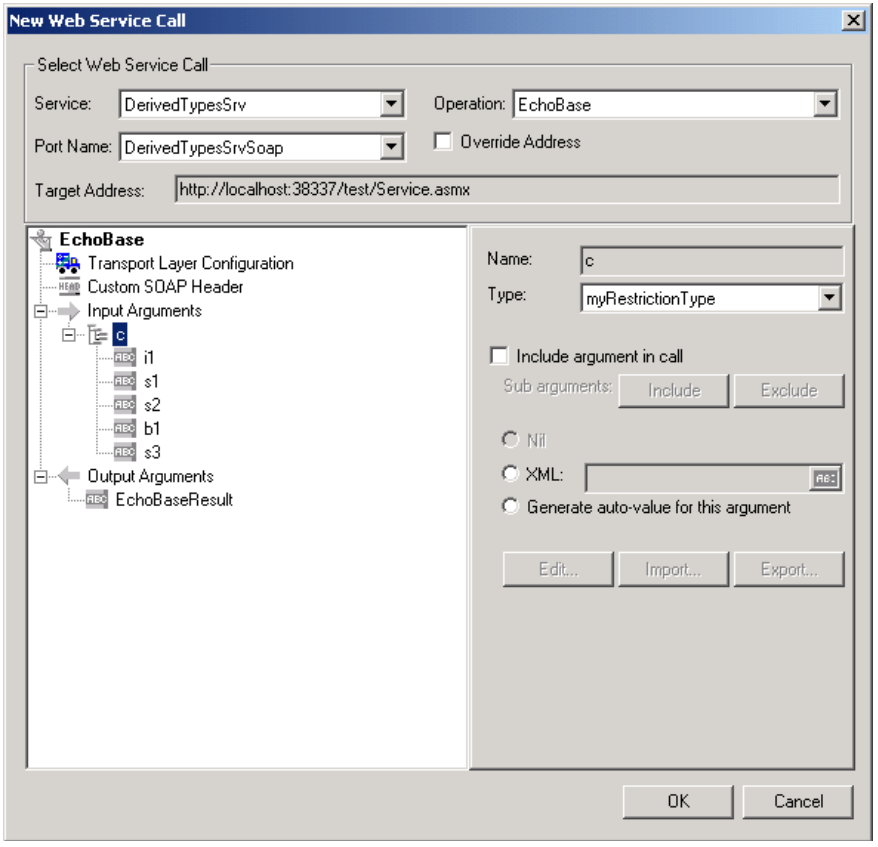
When you click **Add** in the Array Elements section, the Add Array Elements dialog box opens as described below.



- **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.

Derived Types

VuGen supports WSDLs with derived types. When setting the properties for a Web Service Call, you can set the arguments to use the base type or derived type.



After you select the desired type, VuGen updates the argument tree node to reflect the new type.

Abstract Types

Abstract is a declaration type declared by the programmer. When an element or type is declared to be **abstract**, it cannot be used in an instance document. Instead, a member of the element's substitution group, provided by the XML schema, must appear in the instance document. In such a case, all instances of that element must use the **xsi:type** to indicate a derived type that is not abstract.

When VuGen encounters an Abstract type, it cannot create an abstract class and replay will fail. In this case, VuGen displays a warning message beneath the **Type** box, instructing you to replace the Abstract type with a derived type.

Working with Optional Parameters

If your WSDL file contains optional parameters, you can indicate whether or not to include them in the SOAP request.

In WSDL files, optional parameters are defined by one of the following attributes:

minoccurs='0'

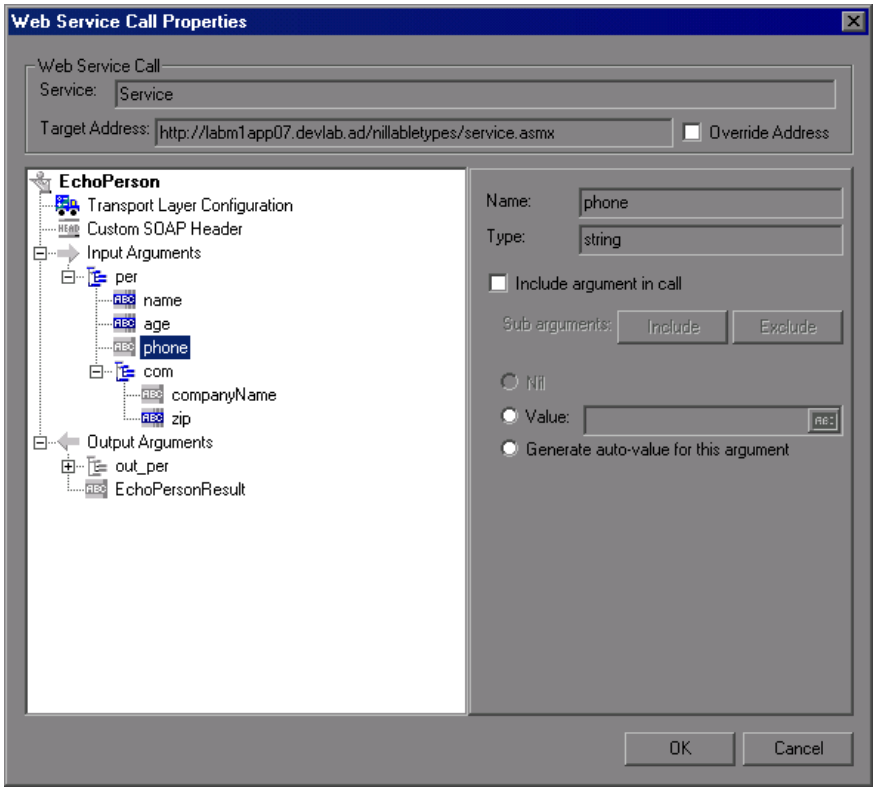
nillable='true'

minoccurs = 0 indicates a truly optional element, that can be omitted. Nillable means that the element can be present without its normal content, provided that the nillable attribute is set to true or 1. By default, the **minoccurs** and **maxoccurs** attributes are set to 1.

In the following example, **name** is mandatory, **age** is optional, and **phone** is nillable.

```
<s:element minOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" name="age" type="s:int" />
<s:element minOccurs="1" name="phone" nillable="true" type="s:string" />
```

When setting argument values for your service call, VuGen indicates the type of element by enabling or disabling the options:



The following table indicates the availability of the options:

Parameter type	Nil radio button	Include arguments in call
Mandatory	disabled	disabled
MinOccurs=0	disabled	enabled
Nillable	enabled	disabled

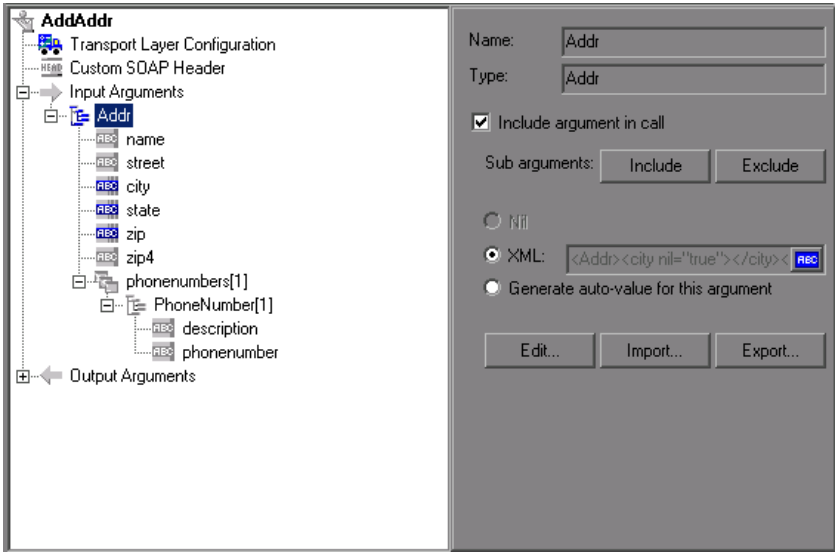
To include a specific optional argument in the service call, click the node and select **Include Argument in Call**. The nodes for all included arguments are colored in blue. Arguments that are not included are colored in gray.

If you include an element on a parent level, it automatically includes all mandatory and nillable children elements beneath it. If it is a child element, then it automatically includes the parent element and all other mandatory or nillable elements on that level. If you specify **Generate auto-value** to a parent element, VuGen provides values for those child elements that are included beneath the parent.

Note: VuGen interprets whether elements are mandatory or optional through the toolkit implementation. This may not always be consistent with the element's attributes in the WSDL file.

To include a sub elements:

1. To include a specific sub element, select it in the left pane and select the **Include Argument in Call** option.
2. To include all sub elements of a parent element, apply **Include Argument in Call** to the parent element and click the **Include** button beneath it.
3. To exclude all sub elements, select the parent element and click the **Exclude** button. It will exclude all omittable arguments.

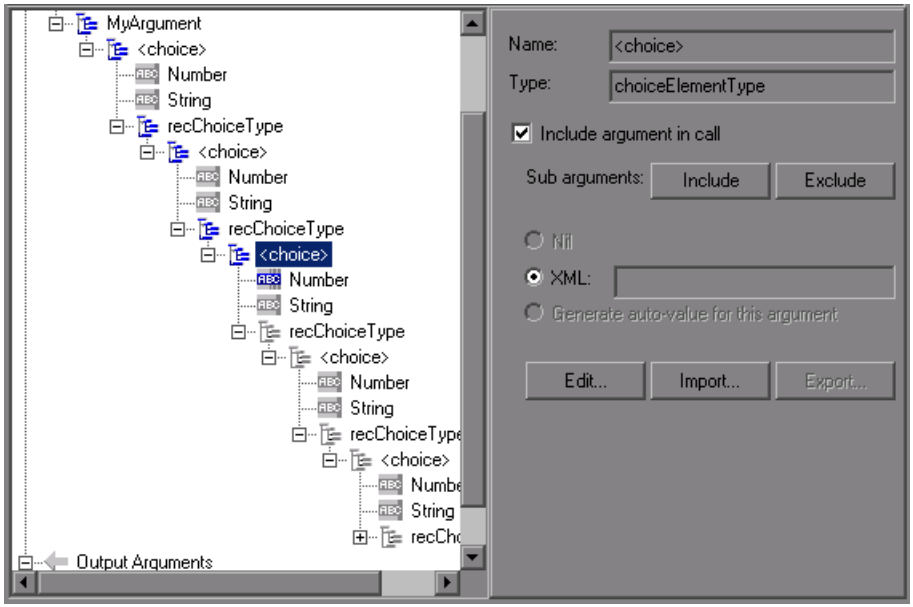


Recursive Elements

Using the Properties dialog box, you can control the level of recursive elements to include in the Web Service call.

To exclude a certain level and exclude those below, highlight the lowest parent node that you want to include and select **Include Argument in Call**. VuGen includes the selected nodes, its mandatory children, and all of its parent nodes.

In the following example, three levels of the Choice argument are included—the rest are not. A non-included node is grayed out.



Choice Optional Elements

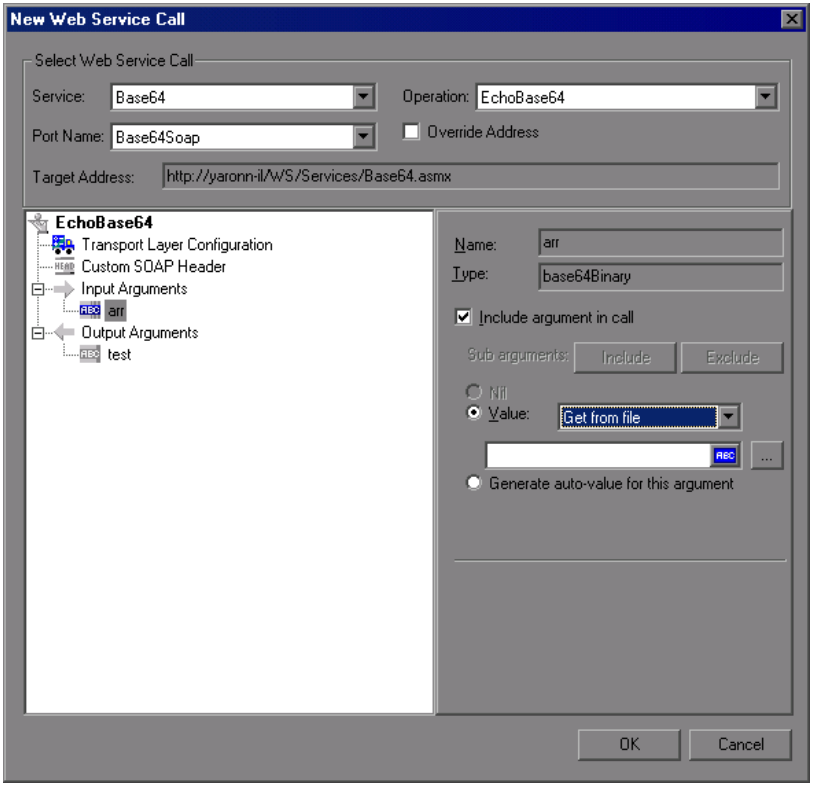
A Choice element in a WSDL defines a set of elements where only one of them appears in the SOAP message. In some cases, one of the Choice elements is optional, while the others are not. In Service Test, you can select the Choice element and still prevent its optional element from appearing in the SOAP envelope. In Tree view, select the Choice element, and clear the **Include argument in call** option. In Script view, delete the line that defines the Choice argument.

Base 64 Encoding

Base 64 encoding is an encoding method used to represent binary data as ASCII text. Since SOAP envelopes are plain text, you can use this encoding to represent binary data as text within SOAP envelopes.

When VuGen detects a WSDL element of **base64Binary** type, it lets you provide an encoded value. You can specify a value in two ways:

- **Get from file.** Reference a file name.
- **Embed encoded text.** Specify the text to encode.

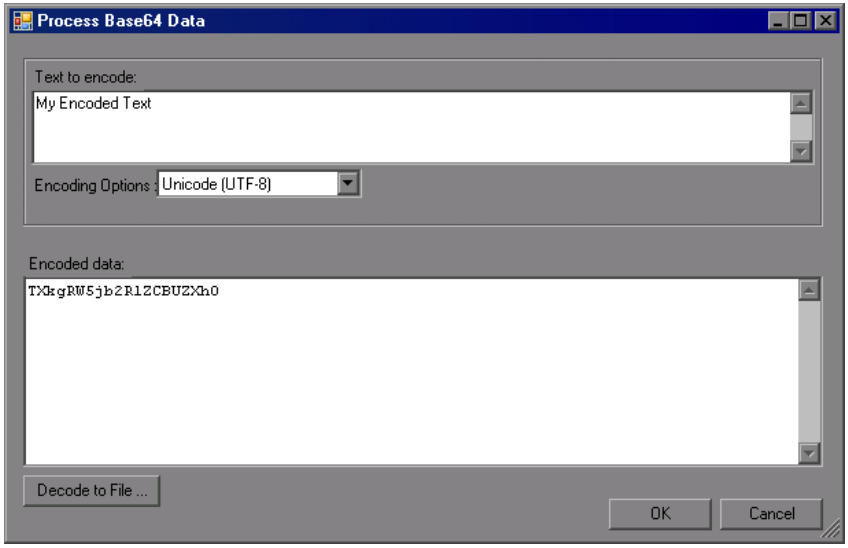


To specify a base64Binary value:

1 Select the **Value** option.

2 To specify a file, select **Get from file** and locate the file using the Browse button below.

3 To specify text, select the **Embed encoded text** option and click the Browse button below. The Process Base64 Data dialog box opens.



Enter text in the **Text to encode** box.

To use an encoding other than the default UTF-8 method, select it from the **Encoding Options** list.

Click **Encode**.

4 Click **OK**. VuGen adds the Web Service call to the script. You can now view the step and its properties in Tree view.

If you referenced the value from a file, the Web Service call will contain the file name:

```
"xml:arr="
    "<arr base64Mode=\"file\">C:\\Load_testing\\TEcho.xml</arr>",
```

If you inserted the actual text using the **Base 64 Encoding Text** option, then the Web Service call in the script will contain the encoded text.

```
"xml:arr="
    "<arr base64Mode=\"encoded\">YWJjZGVmZw==</arr>",
```

Setting a Parameter Value for Base64Binary Data

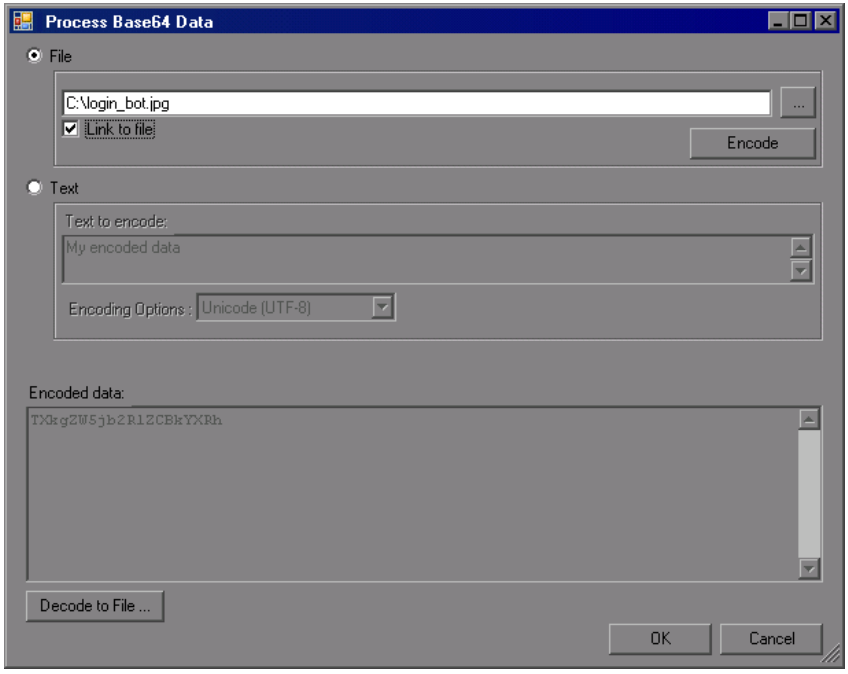
To set the Base64 argument value to a parameter, create a new parameter of of File type, or XML type for Complex arguments. For more information, see "Setting Properties for XML Parameters" on page 1091.

For complex type arguments containing base64Binary values, VuGen lets you process the base64Binary for setting parameter, checkpoint, or emulation values. When specifying the values, you can get values from a **File** or specify the **Text** manually and apply encoding.

If you choose to get the value from a file, specify one of the following options:

➤ **Link to file.** Reference the file containing the values.

➤ **Do not Link to a file.** Use the content of the specified file. VuGen copies the content to the script folder. To use this option, clear the **Link to file** check box.



Tip: It is generally recommended to link to a file since this improves the script's performance. If your text exceeds 10KB, you must link to a file.

To open the Process Base64 dialog box:

1. Create a new parameter for a complex argument.
2. In the grid view (XML parameters or checkpoints), click the Browse button to the right of the value box. The Process Base64 dialog box opens.
3. Specify **File** or **Text** as a source for the value.
4. If you chose **File** as the Source, specify whether or not to link to a file.
5. To decode an encrypted value, for example, a value obtained during replay, click **Decode to File**. For more information, see below.

This section applies to all of the places within VuGen that use the grid view of argument values: the parameter list and checkpoints.

Decoding to a File

VuGen lets you decode the encoded text to a file. This is especially useful for checking the correctness of base64 encoded values returned from the server, such as images.

The following procedure describes how to check if the Record and Replay images match one another.

To validate images using decoding:

1. Create a New Web Service call.
2. Set a value for the Base64 argument, using the **Get from file** option. Specify an image file. Continue creating the script.
3. Save the script and replay it.
4. Switch to the Checkpoint tab and load the Replay values.
5. Click on the Replay value of the Base 64 argument and open its properties.
6. Click **Decode to file**. Specify a file name to which to save the file. Use the same extension as the original file.
7. Compare the decoded image to your original one to verify a match.

Attachments

When transferring binary files such as images over SOAP, the data must be serialized into XML. Serialization and deserialization can cause a significant amount of overhead. Therefore, it is common to send large binary files using an attachments mechanism. This keeps the binary data intact, reducing the parsing overhead.

Using attachments, the original data is sent outside the SOAP envelope, eliminating the need to serialize the data into XML and making the transfer of the data more efficient.

The formats used for passing a SOAP message together with binary data are MIME (Multipurpose Internet Mail Extensions) and the newer, more efficient DIME (Direct Internet Message Encapsulation) specifications. VuGensupports DIME for all toolkits, but MIME only for the Axis toolkit. To use MIME attachments for the .NET toolkit, see "Including MIME Attachments" on page 429.

VuGen 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

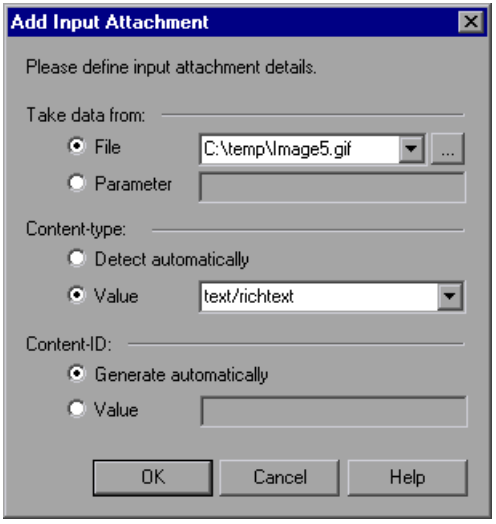
Input Attachments

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)**. VuGen 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, VuGen searches for the attachment file in the script's folder. To add it to the script's folder, select **File** > **Add Files to Script** and specify the file name.
  - **Parameter.** 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 VuGen to automatically determine the content type. You can also select 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, VuGen generates this automatically by VuGen and serves as a unique identifier for the attachment. Optionally, you can specify another ID in the **Value** box.

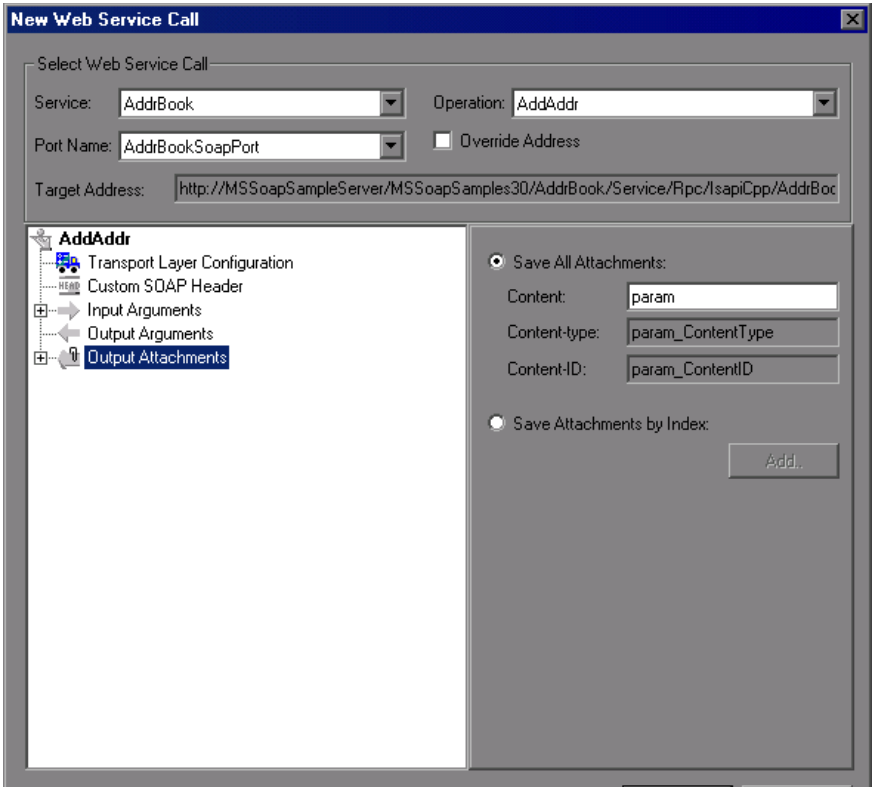
Output Attachments

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)**. VuGen 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,** VuGen creates three parameters for each attachment based on the parameter name that your specify: a parameter containing the attachment data, the content type of the attachment, and a unique ID for the attachment.

For example, if you specify the name MyParam in the Content field, the parameter names for the first attachment would be:

```
MyParam_1
MyParam_1_ContentType
MyParam_1_ContentID
```

When you specify **Save Attachments by Index**, you specify the index number and name of the parameter in which to store the attachment. The parameter name that you specify for **Content**, is used as a prefix for the Content type and Content ID parameters.

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 326.

Working with the XML

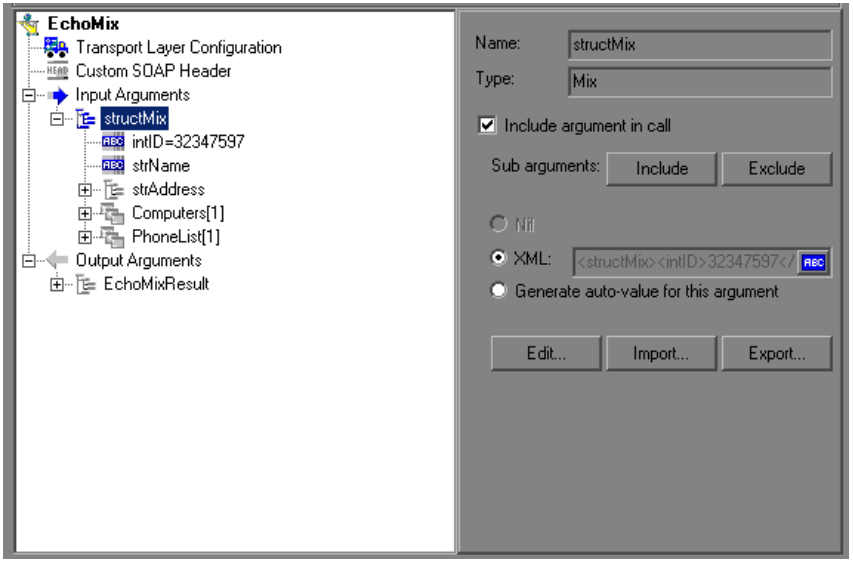
Web Services allow you to viewand edit your XML.

The following sections describe:

- Editing an XML Tree
- Saving and Copying the SOAP Response

Editing an XML Tree

You can use VuGen's XML Editor to view and edit the XML representation of complex types (structures, objects, etc.) and arrays.

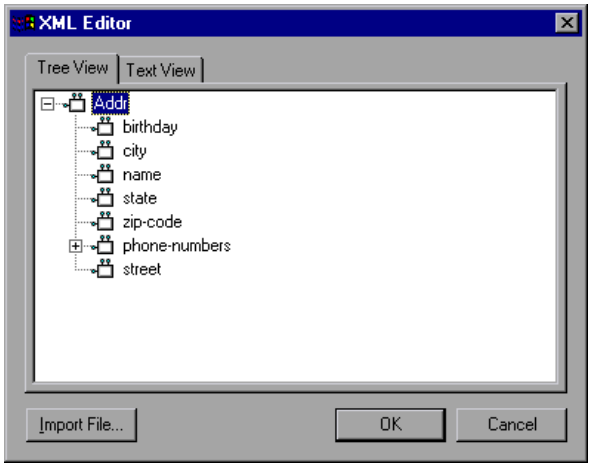


Entering the values for the XML elements is a tedious and error-prone task. VuGen provides you with an interface that simplifies the task of entering, saving, and restoring the information. Once you enter the data manually, you can save it to an XML file using the **Export** option. For subsequent tests, you can import this file without needing to reenter the values a second time.

To edit XML strings:

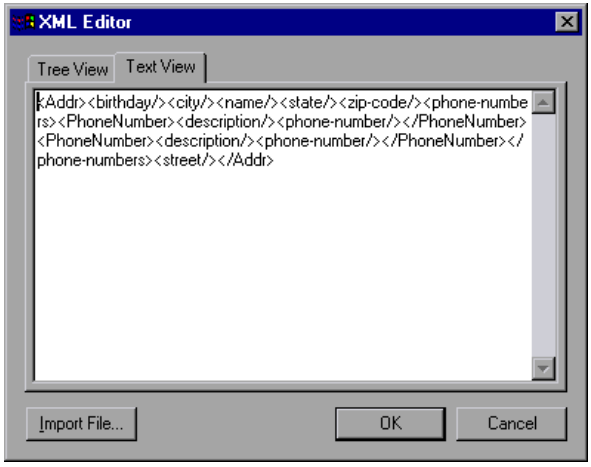
1. Select the Web Service call whose element you want to modify and click the **Step Properties** tab.
2. In the method's tree hierarchy, click on a complex type or array argument. The right-most pane shows the XML code as a single string. Select the **XML** option.
3. To edit the XML code for that entry, click **Edit**. VuGen 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.

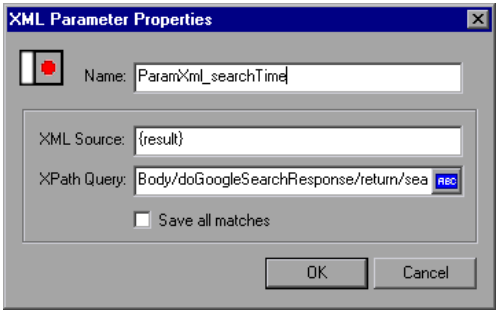
Saving and Copying the SOAP Response

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, select **Copy XML** from the right-click menu.

18

Web Services - Preparing for Replay

After you create a Web Services script, you prepare it for replay so that it can accurately emulate your environment. After replay, you view the test results to see whether the services performed as expected.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Preparing Web Services Scripts for Replay

After you create a script with Web Service calls, you prepare it for replay.

You can enhance it with custom error and log messages or with transactions. See Chapter 6, "Enhancing Vuser Scripts" for more information.

In addition, you can enhance your script with JMS functions, **jms_<suffix>** or XML functions, **lr_xml_<suffix>**. For more information, see the *Online Function Reference* (**Help > Function Reference**).

You can configure run-time settings to help you emulate real users more accurately. These settings include general run-time settings (iteration, log, think time, and general information). Web Services specific settings relate to the JMS transport method, and are described in "Web Services JMS Run-Time Settings" on page 331.

For more information about the general run-time settings, see Chapter 79, "Configuring Run-Time Settings."

Before you replay the script, you can set up checkpoints to make sure that you are getting the correct response from the server. For more information, see below.

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. The **Checkpoint** tab is a central point for defining the required response values for your test.

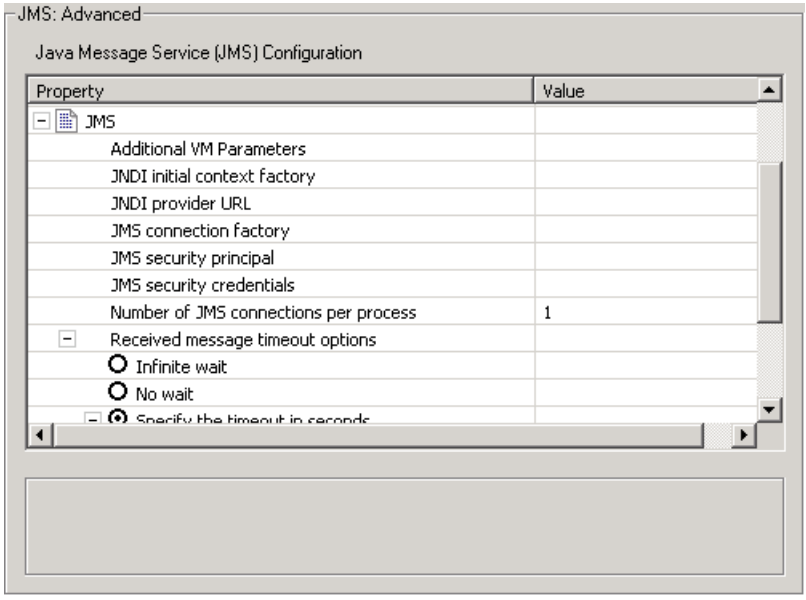
Checkpoint validation is only avaialbe with HP Service Test or HP LoadRunner with a Service Test license. For more informaion, contact HP support.

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. For information on setting the transport level, see Chapter 22, "Web Services - Transport Layers and Customizations."

VuGen lets you configure some of those resources through 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 VuGen.
- **JVM Home.** The location of the external JVM. This should point to the JDK home directory, defined by JDK_HOME. VuGen 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. Select 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 fewer 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.
  - **Specify the timeout in seconds.** Manually specify a timeout value for the message. If the timeout expired and no message has arrived, the operation fails. (default)

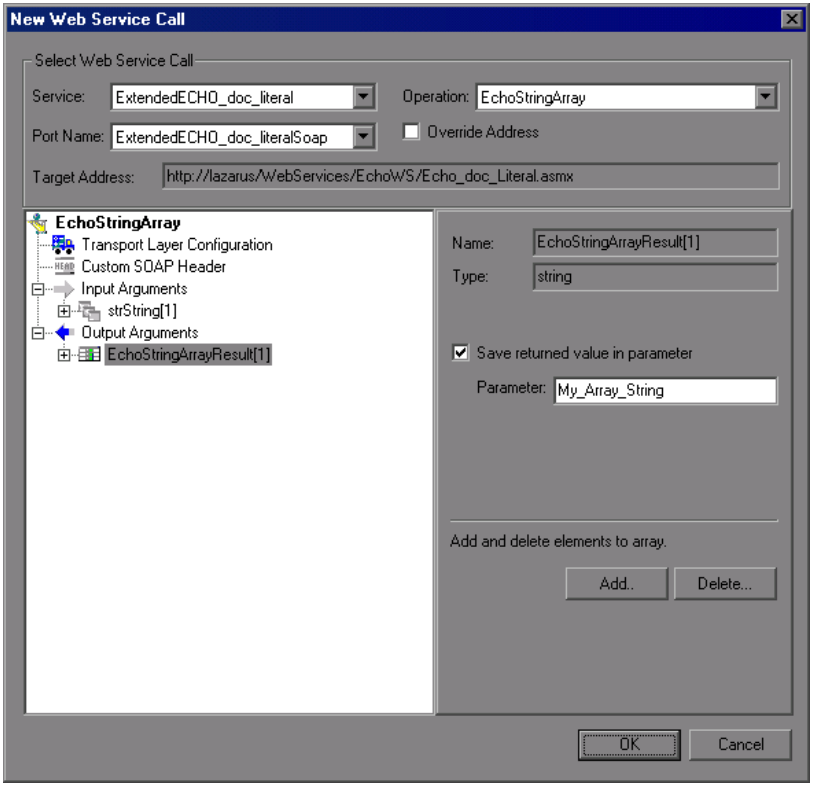
User defined timeout. Specify the amount of seconds to wait for the message before timing out. The default is twenty seconds.

- **Automatically generate selector.** Generates a selector for the response message with the correlation ID of the request (**No** by default). Each JMS message sent to the server has a specific ID. Enable this option if you want VuGen to automatically create a selector that includes the message ID.

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**.

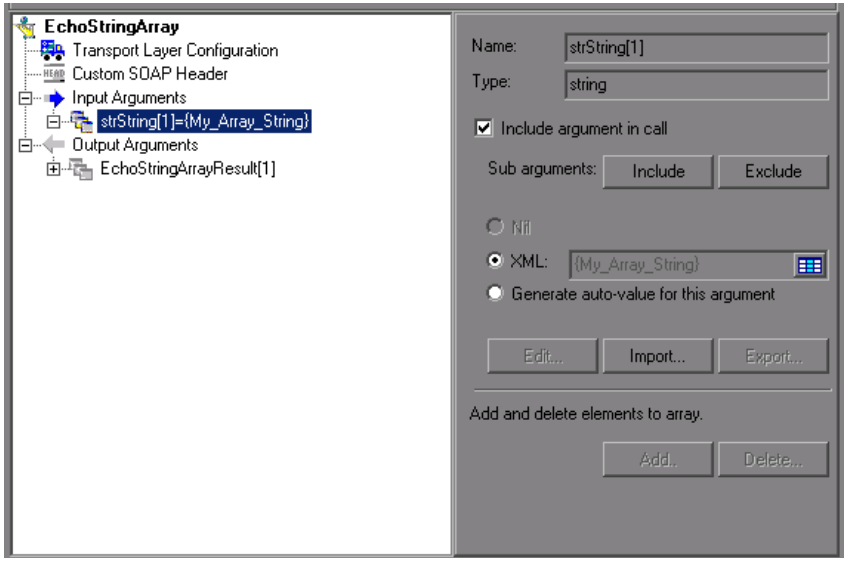


The script shows the saved output parameter as a result argument:

```
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 336.

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 337.

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 the *Online Function Reference*. .

To save an output parameter:

1 View the script in Tree view.

Make sure you are in Tree view. Otherwise, select **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 step'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 the *Online Function Reference*. .

To use a saved parameter for input:

1 View the step 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.

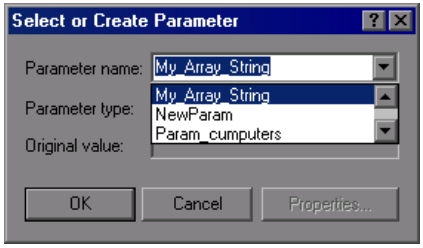
In the left pane, select the input argument whose value you want to replace with a previously saved output parameter.

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 Select 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. Select 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.

Handling Special Cases

This section provides guidelines for running scripts in special cases.

Any Type XSD

For Web Services that have an XSD schema with an **Any** type element, <xsd:element name="*<Any_element>*" type="xsd:anyType" />, make sure your script conforms with the following model:

```
BEGIN_ARGUMENTS,
                "xml:Any_element="
                        "<Any_element>"
                                "<string>the string to send</string>"
                        "</Any_element>",
END_ARGUMENTS,
```

The actual SOAP may differ slightly, but as long as your script conforms to the above model, it will run properly.

You can also send complex type elements for the <any> type. For example:

```
"xml:Any_element="
                        "<Any_element>"
                                "<myComplexTypeName>"
                                        "<property1>123</property1>"
                                        "<property2>456</property2>"
                                "</myComplexTypeName>"
                        "</Any_element>",
```

19

Web Services - Database Integration

Using VuGen, you can integrate with a live database service to retrieve data for your test.

This chapter includes:

About Database Integration

When testing your Web Service, it is vital that you use data that is accurate and up to date. If you use a snapshot of data from a past date, it may no longer be valid or relevant.

The database integration allows you to access values in a database during your test, ensuring that the data is up to date.

The database integration is useful in the following scenarios:

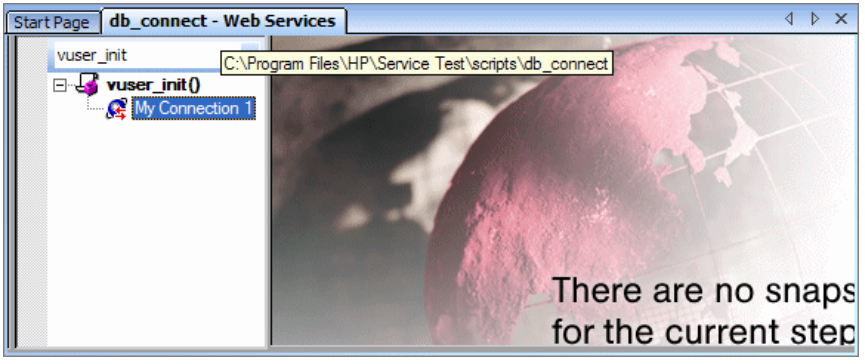
- Using Data Retrieved from SQL Queries
- Validating Database Values after a Web Service Call
- Checking Returned Values Through a Database

Service Test saves all of the database interaction information and displays it in the Test Results report. See Chapter 10, "Viewing Test Results" for more information.

Connecting to a Database

To connect to a database, you add a connection step to your script. Use the Add Step dialog box (**Insert > New Step**) in Tree view, to add a Database Connection step. A built-in Connection String Generator guides you in creating a database connection string specific to your database and credentials. You can also test your connection before inserting the step.

When running your script with iterations, virtual users only repeat the **Action** section of the script. If you include the database connection step in the **Action** section, the test will repeat it for each iteration. Virtual Users only repeat the **Action** section of the script, but not the **vuser_init** or **vuser_end** sections. Therefore, we recommend that you place the database connection step in the **vuser_init** section, and the disconnect step, **lr_db_disconnect** in the **vuser_end** section.



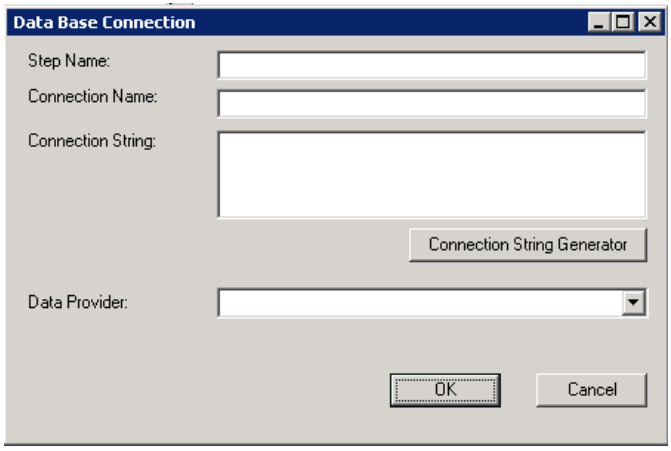
In cases where you only need to do one query and scroll through the data, you should also place the **Database: Execute SQL Query** step in the **vuser_init** section.

To add a database connection step through Tree view:

1 Select **View > Tree View** to enter Tree view (if it is not already visible).

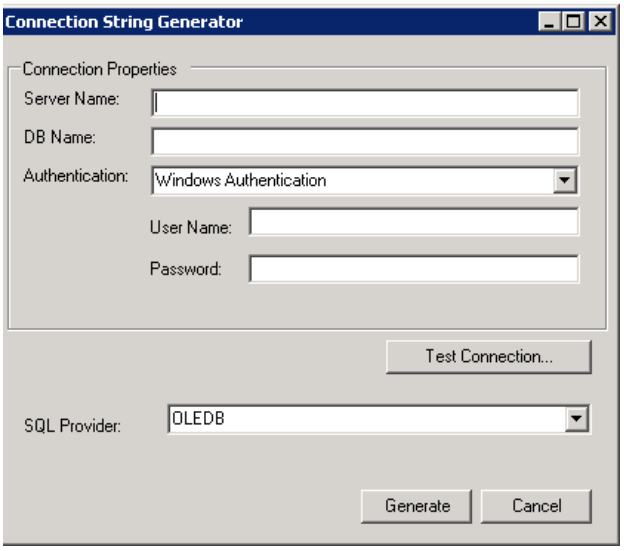
2 Select the desired section: **vuser_init** or **Action**. We recommend placing the connection step in the **vuser_init** section. For more information, see below.

3 Select **Insert** > **New Step**. Choose the **Database: Connect** step. The Database Connection dialog box opens.



4 Specify a **Step Name**, **Connection Name**, and **Data Provider**, OLEDB or SQL.

5 Click **Connection String Generator** to generate a database connection string specific to your environment.



6 Indicate the connection properties:

- **Server Name**
- **Database Name**
- **Authentication** method: Windows Authentication or User/password.
- **Username** and **Password**

7 Click **Test Connection** to verify that the information you provided is correct.

8 Select an **SQL Provider**, OLEDB or SQL, and click **Generate**.

For more information about the required syntax of **lr_db_connect**, see the Online Reference (**Help** > **Function Reference**).

Using Data Retrieved from SQL Queries

In this scenario, the test fetches data from the database and uses it at a later point in the script, such as calls to the Web Service. Since the script retrieves the data during each test run, the data is up to date and relevant.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Web Service call	web_service_call with {<param_name>}
Disconnect from database	lr_db_disconnect

You can iterate through the results in two ways:

- save them to a simple parameter during each iteration
- use VuGen built-in iterations to scroll through the data

For more information, see the Online Reference (**Help > Function Reference**).

In the following example, the **vuser_init** section connects to the database and performs a database query.

```
vuser_init()
{
lr_db_connect("StepName=myStep",
     "ConnectionString=Initial Catalog=MyDB;Data Source=mylab.net;user id =sa
;password = 12345;" ,
     "ConnectionName=MyConnection",
     "ConnectionType=SQL",
     LAST);

lr_db_executeSQLStatement("StepName=MyStep",
          "ConnectionName=MyConnection",
          "SQLQuery=SELECT * FROM Addresses",
          "DatasetName=ds1",
          LAST);

     return 0;
}
```

At the end of your test, disconnect from the database in the **vuser_end** section.

```
vuser_end()
{

     lr_db_connect("StepName=myStep",
          "ConnectionString=Initial Catalog=MyDB;Data Source=LAB1.devlab.net;user id
=sa ;password = soarnd1314;" ,
          "ConnectionName=MyConnection",
          "ConnectionType=SQL",
          LAST);

     return 0;
}
```

In the Action section, you include the steps to repeat. Note the use of the **Row** argument. In the first call to the database, you specify the first row with **Row=next**. To retrieve another value in the same row, use **current**.

```
Action()
{
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=next",
        "OutParam=nameParam",
        LAST);

    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=city",
        "Row=current",
        "OutParam=cityParam",
        LAST);

/* Use the values that you retrieved from the database in your Web Service call */
    web_service_call( "StepName=EchoAddr_101",
        "SOAPMethod=SanityService|SanityServiceSoap|EchoAddr",
        "ResponseParam=response",
        "Service=SanityService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227168459.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>{nameParam}</name>"
                "<street></street>"
                "<city>{cityParam}</city>"
                "<state></state>"
                "<zip></zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

return 0;
}
```

Validating Database Values after a Web Service Call

In this scenario, the test executes a Web Service call that modifies a database on the backend. The goal of this scenario is to validate that the resulting values in the database are correct.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Check the data	lr_checkpoint
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

For more information, see the Online Reference (**Help > Function Reference**).

The following example illustrates this process of checking the data:

```
Action()
{
/* A Web Service call that modifies a database on the back end. */
    web_service_call( "StepName=addAddr_102",
        "SOAPMethod=Axis2AddrBookService|Axis2AddrBookPort|addAddr",
        "ResponseParam=response",
        "Service=Axis2AddrBookService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227169681.inf",
        BEGIN_ARGUMENTS,
        "xml:arg0="
            "<arg0>"
                "<name>{Customers}</name>"
                "<city>{City}</city>"
            "</arg0>",
        END_ARGUMENTS,
        LAST);

/* Query the database by the cusotmer name that was modified by the Web Service*/
    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = '{Customers}' ",
        "DatasetName=ds1",
        LAST);

/* Get the values retrieved by the database query. */
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);

/* Compare the actual value with the expected value stored in the database. */
    lr_checkpoint("StepName=validateCustomer",
        "ActualValue={Customers}",
        "ExpectedValue={CustomerName}",
        "Compare=Equals",
        "StopOnValidationError=false",
        LAST);

return 0;
}
```

Checking Returned Values Through a Database

In this scenario, the user executes a Web Service call which returns an XML response. The goal of this scenario is to validate the response of the Web Service call against expected values. The expected values are stored in a database. The script fetches the expected results from a database and then compares them with the actual response.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call with **Result=<result_param>**
Execute an SQL query	lr_db_executeSQLStatement
Retrieve the expected data	lr_db_getvalue to <param_name>
Validate the data	soa_xml_validate with an XPATH checkpoints.
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

You can use the XML validation tool to create a checkpoint for the response data. When creating the validation step, use the database parameter that you retrieved through **lr_db_getvalue**.

The following example illustrates a typical validation of data returned by a Web Service call. The Validation step compares the actual expected results:

```
Action()
{
    web_service_call( "StepName=GetAddr_102",
        "SOAPMethod=AddrBook|AddrBookSoapPort|GetAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227172583.inf",
        BEGIN_ARGUMENTS,
        "Name=abcde",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = 'abcde' ",
        "DatasetName=ds1",
        LAST);

    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);

    soa_xml_validate ("StepName=XmlValidation_1146894916",
        "Snapshot=t623713af7a594db2b5fef43da68ad59d.inf",
        "XML={GetAddrAllArgsParam}",
        "StopOnValidationError=0",
        BEGIN_CHECKPOINTS,
            CHECKPOINT,"XPATH=/*[local-name(.)='GetAddr'][1]/*[local-
name(.)='Result'][1]/*[local-name(.)='name'][1]","Value_Equals={CustomerName}",
        END_CHECKPOINTS,
        LAST);
    return 0;
}
```

For more information, see the Online Reference (**Help** > **Function Reference**).

Performing Actions on Datasets

VuGen lets you perform actions on datasets returned by SQL queries.

The **lr_db_dataset_action** function performs the following actions on datasets:

- **Reset.** Set the cursor to the first record of the dataset.
- **Remove.** Releases the memory allocated for the dataset.
- **Print.** Prints the contents of the entire dataset to the Replay Log and other test report summaries.

Note that when you retrieve binary data through **lr_db_getvalue**, you cannot print its contents using the **Print** action.

For information about the syntax and usage of this function, see the Online Reference (**Help** > **Function Reference**).

20

Web Services - Security

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.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Adding Security for Web Service Testing

When building Web Service applications, there is a challenge in building scalable applications that are secure. You can secure Web Services by having the message sent over a secure transport, such as Secure Sockets Layer (SSL), but this is limited to point-to-point communication.

To allow you to send your messages securely, VuGen supports several security mechanisms, Security Tokens (WS-Security), and SAML.

For more information on tokens, see below. For more information on SAML, see "Setting SAML Options" on page 360.

To learn more about customizing WS-Security and testing with WCF, see Chapter 21, "Web Services - Advanced Security and WS Specifications."

Note: If your WSDL is located in a secure location, you must provide the security information through the Manage Services dialog box. For more information, see "Specifying WSDL Connection Settings" on page 258.

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Legacy security model using **web_service_set_security**. For information on the Scenario model, see Chapter 21, "Web Services - Advanced Security and WS Specifications."

The following table lists the considerations for using each of the models.

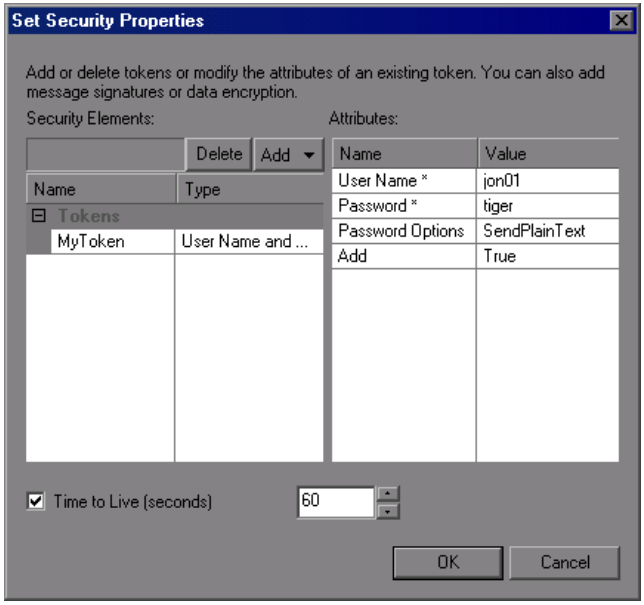
Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework, such as Axis2 or Metro (WSIT)
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust
You are having trouble using the new model or find the capabilities of the legacy more adequate for your needs	You are having trouble using the legacy model or you find the capabilities of the new model more adequate

Adding Security to a Web Service Call - a General Workflow

The following section describes the general workflow for adding security to a Web Service call:

To add Web Service security:

1. Place the cursor at the point at which you want to add the security settings. In most cases, we recommend that you place it in the vuser_init section so that the security scope will be applied to the whole script. If you only want the security for specific calls, place it at the desired location.
2. Select **Insert** > **New Step** to open the Add Step dialog box.
3. Select **Web 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 Select a token type. Common tokens are Username and X.509 certificate. For information about the token types, see "Security Tokens and Encryption" on page 355.

In the **Logical Name** box, assign an arbitrary name for the token to be used by VuGen in identifying the token.

Add any relevant information, such as **User Name** and **Password** for the User Name and Password type token.

To send the token explicitly in the SOAP envelope header, select **True**. To exclude the token from the SOAP envelope header, select **False**.

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 **Add** to add a message signature and encryption if they are required. Both signatures and encryptions require you to specify a token previously defined as the encrypting/signing token. See the "Examples Using web_service_set_security" on page 363 to learn how to encrypt or sign a specific XPath in the SOAP.

8 To cancel the security settings at a specific point within the script, add a **Web Service Cancel Security** step at the desired point.

9 Click **OK**. VuGen inserts a Web Services Set Security step at the location of the cursor.

"Examples Using web_service_set_security" on page 363 illustrates some of the common security settings.

Security Tokens and Encryption

The WS-Security specification lets you place security credentials in the actual SOAP message. You accomplish this by instructing a client to obtain security credentials from a source that is trusted by both the sender and receiver. When a SOAP message sender sends a request, those security credentials, known as security **tokens**, are placed in the SOAP message. When the Web server receives the SOAP request, it does not need to send additional requests to verify the integrity of the sender. The server verifies that the credentials are authentic before letting the Web Service execute the application. By not having to go back to the source of the credentials, this significantly improves the application's scalability.

To further secure Web Services, it is common to use digital signatures or encryption for the SOAP messages. Digitally signing a SOAP message verifies that the message has not been altered during transmission. Encrypting a SOAP message helps secure a Web Service by making it difficult for anyone other than the intended recipient to read the contents of the message.

The Web Services security mechanism associates security tokens with messages. This mechanism supports several security token formats to accommodate a variety of authentication requirements. For example, a client might need to provide a proof of identity or a security certificate.

To support WS-Security, VuGen allows you to create security tokens for your script. You can create multiple tokens and set their properties. After creating a token, you use it to sign or encrypt a SOAP message.

In certain instances, you do not send the token explicitly—you use the token for the purpose of signatures or encryption, without including the actual token in the SOAP envelope header. Using the **Add** option, you can indicate whether to send the actual token explicitly.

The available tokens are **Username and Password**, **X.509 Certificate**, **Kerberos Ticket**, **Kerberos2 Ticket**, **Security Context Token**, and **Derived Token**. The information you need to provide differs for each token.

- **User Name and Password.** The **User Name and Password** token contains user identification information for the purpose of authentication: **User Name** and **Password**.

You can also specify Password Options, indicating how to send the password to the server for authentication: **SendPlainText**, **SendNone**, or **SendHashed**.

- **X.509 Certificate.** This security token is a token based on an X.509 certificate. To obtain a certificate, you can either purchase it from a certificate authority, such as VeriSign, Inc. or set up your own certificate service to issue a certificate. Most Windows servers support the public key infrastructure (PKI) which enable you to create certificates. You can then have it signed by a certificate authority or use an unsigned certificate.

When you add an X.509 token to the Vuser script, you specify the **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.

VuGen 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, see the *Online Function Reference* (**Help** > **Function Reference**).

Adding the Security Policy

To add a security policy to a section of your script, you enclose the relevant steps with **Web Service Set Security** and **Web Service Cancel Security** steps.

When you add a **Web Services Set Security** step to your script, VuGen adds a **web_service_set_security** function that contains arguments with the tokens, message signatures, and encryption that you defined in the security properties.

```
web_service_set_security(
  SECURITY_TOKEN, "Type=USERNAME", "TokenName=mytoekn1",
  "UserName=bob", "Password=123", "PasswordOptions=SendNone",    "Add=True",
LAST);
```

Parameterization is not supported for the following arguments: **Token Type, Logical Name, Base Token, Issuer Token** or **Derive From** arguments.

Working with Message Signatures and Encrypted Data

When you add a security token to a SOAP message, it is added to the SOAP message in the form of an XML element in the WS-Security SOAP header.

The message, however, is exposed and therefore requires additional security. This is especially true when the credentials, including the password, are sent in plain text as it is with role-based security.

The two methods used to secure the data are digital signatures and encryption.

- **Digital Signatures.** Digital Signatures are used by message recipients to verify that messages were not altered since their signing. The digital signature is usually in the form of XML within the SOAP message. The recipient checks the signature to make sure it is valid. Certain environments, such as WSE, automatically verify the signature on the SOAP recipient's computer.

➤ **Encryption.** Although the XML digital signature offers a mechanism for verifying that the message has not been altered since it was signed, it does not encrypt the SOAP message—the message is still plain text in XML format. To secure the message in order that it should not be exposed, you encrypt it, making it difficult for an intruder to view and obtain a user's password.

VuGen 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 SAML Options

VuGen supports SAML (Security Assertion Markup Language) for Web Services. SAML is an XML standard for exchanging security-related information, called **assertions**, between business partners over the Internet. The assertions can include attribute statements, authentication, decision statements, and authorization decision statements.

SAML uses brokered authentication with a security token issued by STS (Security Token Service). The STS is trusted by the client and the Web Service to provide interoperable security tokens. SAML tokens are important for Web Service security because they provide cross-platform interoperability and a means of exchanging information between clients and services that do not reside within a single security domain.

You can set the SAML settings for an entire script or part of the script. 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** step.

Signing an SAML Assertion

VuGen provides a method for signing an unsigned SAML assertion. As input, you provide the unsigned assertion, a certificate file, and the optional password. VuGen provides a signed SAML assertion as output.

To add this method to your script, use the Add Step dialog box (**Insert** > **New Step**). VuGen adds a **ws_sign_saml_assertion** to the script. For syntax information, see the *Online Function Reference* (**Help** > **Function Reference**).

Policy Files

SAML policy files follow the WSE 3.0 standard and define the attribute values for the SAML security. By default, VuGen uses the **samlPolicy.config** file located in the installation's **dat** folder.

When entering SAML security information, you can enter it manually in the properties dialog box, or you can refer to a policy file containing all of the security information. You can create your own policy file based on samlPolicy.config.

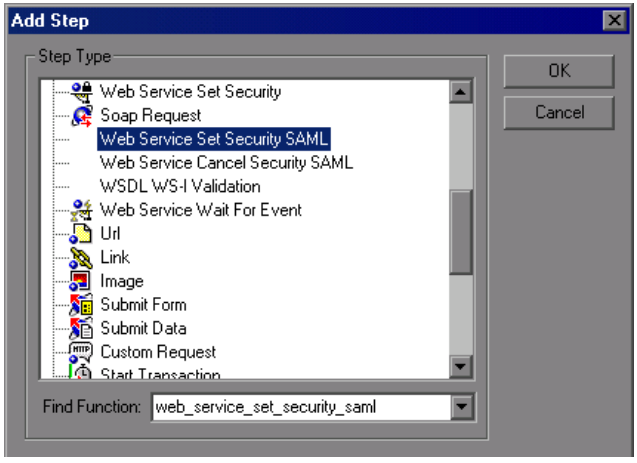
You can modify the policy file to include values for the security parameters, such as username and certificate information. When adding a SAML security step to your script, if you explicitly specify values for the security arguments, they override the values in the policy file.

If you make changes to the default policy file, we recommend that you copy the new policy file to your script's folder. Make sure to save custom policy files with a **.config** extension to insure that they remain with the script, even when running it on other machines or calling it from the LoadRunner Controller.

To learn more about the SAML policy files, see the SAML STS example on the MSDN Web site. If you want to emulate SAML Federation behavior, copy the **samlFederationPolicy.config** file from the data folder to your script's folder, and specify it as the policy file.

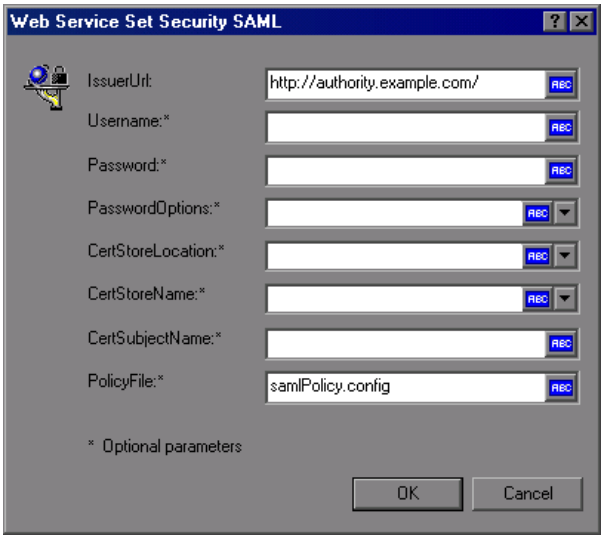
To add SAML security:

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. Select **Insert** > **New Step** to open the Add Step dialog box.



3 To add SAML security, select **Web Services Set Security SAML**.

Enter the desired information. If you enter values into this dialog box, they override any values 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, select **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).

Examples Using web_service_set_security

This section illustrates several common security scenarios.

Authenticating with a Username Token

The following example illustrates the sending of a message level username/password token (a username token), where the user name is John and the password is 1234.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        LAST);
```

Signing with an X.509 Certificate

The following example shows a script using an X.509 certificate for a digital signature.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        LAST);
```

Note that your certificate needs to be installed in the Windows certificate store. In the example above, you need to set the actual store name, store location, and subject name of your certificate.

Signing a Specific Element with an X.509 Certificate

It is possible to sign only a specific element in a message. The following example signs a specific element using an XPATH expression:

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert", "TargetPath=//*[local-
name(.)='someElement' and namespace-uri(.)='http://myNamespace']",
        LAST);
```

Encrypting with a Certificate

The following sample encrypts a message with the service's X.509 certificate.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
"StoreLocation=CurrentUser", "Add=False",
        ENCRYPTED_DATA, "UseToken=serviceCert",
        LAST);
```

After you specify the details of your X.509 certificate, you can encrypt a specific XPATH in the message.

Since we want to generate a Subject Key Identifier, we set the Add value to False. For more information, see "Using SubjectKeyIdentifier" on page 367.

Authenticating with a Username Token and Encrypting with an X.509 Certificate

The following example sends a username token to the service and encrypts it with the server's X.509 certificate:

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
"StoreLocation=CurrentUser", "Add=True",
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myUser",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        ENCRYPTED_DATA, "UseToken=serviceCert", "TargetToken=myUser",
        LAST);
```

The **UseToken** and **TargetToken** properties indicate which token to use and which to encrypt. Their values reference the **LogicalName** property of the tokens.

Encrypting and Signing a Message

This example shows how to sign a message using a private key and then encrypt it using the service's public key.

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        SECURITY_TOKEN, "Type=X509","LogicalName=serverToken",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serverCert",
"StoreLocation=CurrentUser", "Add=False",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        ENCRYPTED_DATA, "UseToken=serverCert",
    LAST);
```

Referencing an X.509 Certificate Using a Hash

In certain cases, you may be unable to reference a certificate with a subject name. This example shows how to reference the certificate using its unique hash.

```
web_service_set_security(
  SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert", "StoreName=My",
  "IDType=Base64KeyID", "IDValue=pOI0+1iuotKLIO91nhjDg5reEw0=",
  "StoreLocation=CurrentUser", "Add=False",
  ENCRYPTED_DATA, "UseToken=serviceCert",
 LAST);
```

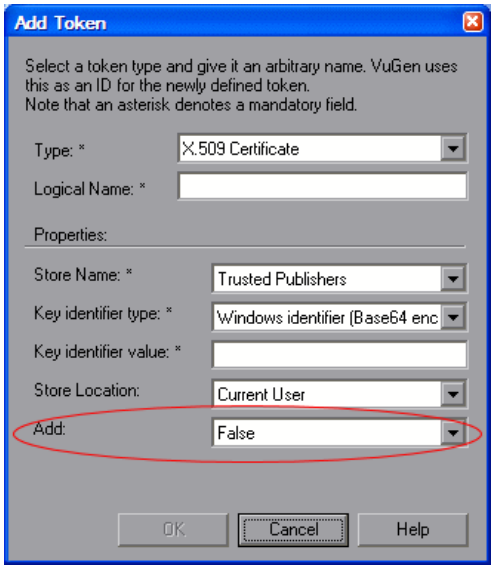
Customizing Your Security

The following sections describe how to configure special cases common to Web Service security.

Using SubjectKeyIdentifier

By default, Service Test adds all of the defined X.509 tokens to the SOAP envelope and references them as binary tokens. It is also possible to exclude the tokens from the message and reference them with a **SubjectKeyIdentifier**. This is common with tokens that are used for encryption.

In order to achieve this, when you add the token through the UI, choose **False** for the Add option.



Alternatively, you can configure this setting in the script:

```
SECURITY_TOKEN, "Type=X509","LogicalName=myToken", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=False",
```

If you are using a SKI (Subject Key Identifier) you may also need to modify the **useRFC3280** settings as described in "Customizing WS-Security" on page 370.

Username Customization

When you add a new username token, the editor shows the **web_service_set_security** as follows:

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=john", "Password=1234", "PasswordOptions=SendPlainText", "Add=True",
        LAST);
```

There are two additional settings that you can use for customization, that are not available in the user interface.

Name	Meaning	Possible values
IsNonceIncluded	Should the username token contain a nonce	True (default) or False
TimestampFormat	Should the username token contain a timestamp. If so, in what format	➤ **None**. no timestamp ➤ **Full**. a <timestamp> element with <created> and <expired> inner elements ➤ **Created**. (default) only a <created> element

Add these options to **web_service_set_security** in the following way:

```
web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=John", "Password=1234",  "PasswordOptions=SendPlainText",
"IsNonceIncluded=true", "TimestampFormat=Full", "Add=True",
        LAST);
```

Customizing Encryption

You can customize encryption by indicating how to encrypt the element—encrypt the whole element or only its content. This is common when encrypting tokens such as a user name.

You can use the following setting to determine the exact encryption type:

Name	Meaning	Possible values
EncryptionType	What to encrypt	➤ Element ➤ Content (default)

Add this option to **web_service_set_security** in the following way:

```
web_service_set_security(
…
ENCRYPTED_DATA, "UseToken=myToken", "TargetToken=myOtherToken",
"EncryptionType=Element",
LAST);
```

Customizing WS-Security

It is sometimes necessary to change the algorithm Service Test uses for encryption or to modify some other low-level security details.

To change either of these items, open the **%Service Test%/bin/mmdrv.exe.config** file in a text editor. If this file does not contain the <**microsoft.web.services2**> element, add it as shown below.

```
<configuration>
…
  <microsoft.web.services2>
    <security>
    <x509 storeLocation="CurrentUser" allowTestRoot="true" useRFC3280="true" />
      <binarySecurityTokenManager valueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3">
        <sessionKeyAlgorithm name="TripleDES" />
        <keyAlgorithm name="RSA15" />
      </binarySecurityTokenManager>
      </security>
  </microsoft.web.services2>
…
<configuration>
```

Set the element values as required:

Name	Meaning	Possible values
verifyTrusy	Whether to check sent/received x.509 certificate's validity	➤ True (default) ➤ False
sessionKeyAlgorithm	The algorithm that the session symmetric key will use to encrypt the message	➤ AES128 ➤ AES192 ➤ AES256 ➤ TripleDES
keyAlgorithm	The algorithm used by the public key to encrypt the session key	➤ RSA15 ➤ RSAOAEP
useRFC3280	Whether to generate subject key identifiers that are interoperable and not windows specific	➤ True ➤ False (default)

21

Web Services - Advanced Security and WS Specifications

Service Test lets you customize your script to support additional WS standards. These include WS standards implemented in WCF and other WS - <*spec_name*> specifications.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Advanced Security and WS Specifications

Service Test allows you to test Web Services that utilize advanced security and WS-Specifications. Such services can be written in various platforms such as WCF (Windows Communication Foundation), Metro (WSIT), and Axis2. For WCF services, Service Test also supports proprietary standards and transports.

You enable this support by setting up a security scenario. Each scenario represents a typical environment used in conjunction with Web Service calls. Service Test provides several built-in security scenarios that are commonly used. It applies the scenario's settings individually to each service.

For the built-in scenarios, the user interface lets you provide identity information where required. You can customize security, transport, proxy, and other advanced settings.

If you cannot find a scenario that corresponds to your environment, you can use the generic custom scenario.

For a "How To" guide on selecting a scenario, see "Tips and Guidelines" on page 397.

Choosing a Security Model

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Scenario security model. The Legacy model refers to the manual addition of **Web Service Set Security** steps, or the **web_service_set_security** function. For information on the Legacy model, see Chapter 20, "Web Services - Security."

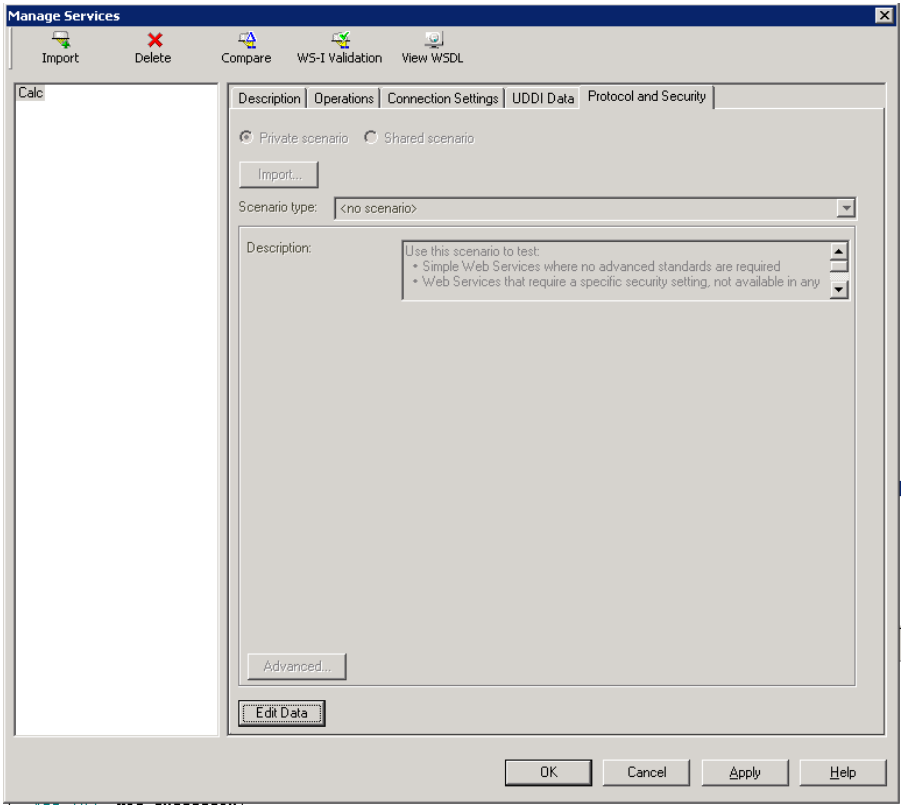
The following table lists the considerations for using each of the models.

Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework such as Axis2 or Metro (WSIT).
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust
You are having trouble using the new model or find the capabilities of the legacy functions adequate	You are having trouble using the legacy model or you find the capabilities of the new model more adequate

Setting the Security Scenario

You assign security scenarios on a service level per script—you assign a different security scenario for each service in your script.

The Manage Services window contains an interface to create and edit security scenarios for individual services. You access this interface from the **Protocols and Security** tab.



Using the Security Scenario editor, you can also create scenario files independent of the script. You can save them to a shared location for personal use or for collaboration with others.

Setting the Security Scenario for a Service

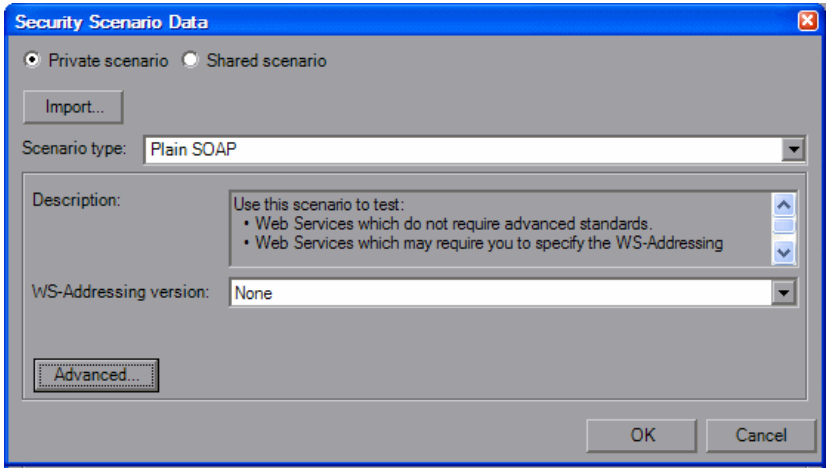
To assign a security scenario to a specific service, use the Manage Services window. The **Protocol and Security** tab contains the interface to create and view security scenarios for individual services.

You can select a scenario in three ways:

- **Private scenario.** Create a new scenario by selecting one of the built-in ones and customizing it for your Web Service.
- **Imported scenario.** Use a scenario created at an earlier time. The scenario will be editable, and if someone modifies the original scenario, it will not affect you.
- **Shared scenario.** Load a security scenario already configured by another user from a remote location or the file system. You cannot edit this scenario's settings from the Manage Services window. If someone edits the scenario, it will affect your environment. You usually use this option after working with the product for some time and saving the scenario files.

To create a security scenario for a specific service:

1. Click **Manage Services**. In the left pane, select the service for which you want to set the security scenario. If necessary, import a service, as described in "Importing Services" on page 253.
2. Select the **Protocol and Security** tab and click the **Edit Data** button. The Security Scenario Data dialog box opens.



3 Load a scenario. When you begin working with security scenarios, use the first method. After you have created several scenarios, you can use the other methods.

a To select a built-on security scenario for the current service, choose **Private scenario**.

In the Scenario type box, choose a scenario. The scenario types are described in "Scenario Types" on page 378.

Specify the required values for your scenario. For details, see "Specifying Scenario Information" on page 380.

b To use an existing scenario with the ability to modify it, choose **Private scenario.** Click **Import**. In the Shared Scenario dialog box, select a stored scenario. If required, modify the settings as described in "Specifying Scenario Information" on page 380.

c To use an existing scenario without the option of changing it, choose **Shared Scenario**. Use the Browse button to open the Shared Scenario dialog box and select a stored scenario.

Click **OK**. Your service is now loaded with a security scenario. If someone modifies the scenario file at its source, it will affect your script.

4 Click **Advanced** to configure the Proxy, Encoding, and other advanced setting (optional). For most scenarios, the default settings are ideal. For information about these settings, see "Advanced Settings" on page 387.

5 Click **OK** to close the dialog box and save your script.

Saving Security Scenarios for Collaboration

One user can customize a security scenario and make it available to others. Using the Security Scenario Editor, you customize your settings and save them to a scenario file.

To create a new security scenario:

1. Choose **SOA Tools** > **Security Scenario Editor**.
2. Select a **Scenario type** and enter the relevant information.
3. When enabled, click **Advanced** to configure the Proxy, Encoding, and other setting as described in "Advanced Settings" on page 387. Click **OK** to close the dialog box.
4. For a new scenario, click **Save as**, and specify a file name and path for the scenario file.

To edit an existing stored scenario:

1. Choose **SOA Tools** > **Security Scenario Editor**.
2. Click the **Open** button and browse for an existing scenario file.
3. Modify the scenario settings as required.
4. Click the **Save** or **Save as** button.
5. When enabled, click **Advanced** to configure the Proxy, Encoding, and other advanced settings. For more information, see "Advanced Settings" on page 387.
6. Click **OK** to close the Security Scenario editor.

Scenario Types

The scenario describes the configuration of your Web Service. It contains information such as encoding, identities, proxy, and so forth. Service Test provides a Security Scenario editor that allows you to configure the settings for each scenario.

To determine the scenario that best fits your service, refer to the table below. If you are unsure which scenario to choose, we recommend that you use the **Custom Binding** scenario which allows you to test any service. For more information, see "Custom Binding" on page 383.

The following table lists the built-in scenarios and describes when to use them.

Scenario Name	When to use
<no scenario>	➤ Default setting: if you have no need for a special security/protocols configuration, leave this value as is. Simple Web Services where no advanced standards are required. ➤ Scripts that use the legacy security model described in Chapter 20, "Web Services - Security." ➤ Web Services that require a specific security setting, not available in any of the existing scenarios ➤ If you select a built-in scenario and experience problems in replay, it is possible that no scenario is required and the problem is elsewhere. Reset the value to **<no scenario>.**
Plain SOAP	➤ Web services which do not require advanced standards ➤ Web services which may require you to specify the WS-Addressing version
MTOM	➤ MTOM enabled Web services ➤ Web Services which may require you to specify the WS-Addressing version

The following table shows the scenarios for Web Services that utilize WCF. The WSHttpBinding-based scenarios are divided according to the way the client authenticates itself to the server. For example, if your client presents a user name and a password to the server, choose the **Username (message protection)** scenario. The user interface lets you provide the identity information in the form of a user name or a certificate as required.

WCF Scenario Name	When to use
WSHttpBinding - No Authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client is not authenticated ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - Windows authentication	➤ Client and server use Windows authentication ➤ Security is based on Kerberos or SPNEGO negotiations ➤ Communication may utilize advanced standards such as secure conversation and MTOM
wsHttpBInding - Certificate authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client uses its own X.509 certificate for signature ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - username (message protection) authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client is authenticated with a username and password ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - username (transport protection) authentication	➤ SSL is enabled ➤ Client is authenticated with a username and password ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSFederationHttpBinding	➤ Client authenticates against the STS using a predefined scenario ➤ Client uses the token given from the STS to authenticate against the server
Custom Binding	➤ Web Service that uses WS-* standards ➤ WCF services of any configuration

Specifying Scenario Information

This section describes the values required for each of the security scenarios.

<no scenario>

Since you are not using a secure scenario, you do not have to specify any values.

Plain SOAP

For this type of scenario, if your service uses WS-Addressing, specify the version.

MTOM

For MTOM type scenarios, if your service uses WS-Addressing, specify the version.

No Authentication

In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 385. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information.

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Windows Authentication

This WCF scenario uses Windows Authentication.

You declare the expected identity of the server in terms of its **SPN** or **UPN** identities. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.

Certificate Authentication

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 385. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username Authentication (Message Protection)

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.

➤ **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 385. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

➤ **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username (Transport Protection) Authentication

This WCF WSHttpBinding scenario enables SSL and authenticates the client with a user name and password on the message level.

Specify the following settings:

➤ **Username. Password.** The client's user name and password credentials.

Federation

In the WSFederationHttpBinding scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server.

Therefore, two bindings are needed, one against the STS and another against the application server.

First, use the Security Scenario editor to define an STS binding. For more information, see "Saving Security Scenarios for Collaboration" on page 377. When setting the binding against the application server, specify this file in the **Referenced file** box.

For the Federation scenario, specify the following server information:

➤ **Transport.** HTTP or HTTPS

➤ **Encoding.** Text or MTOM

For the Federation scenario, specify the following security information:

- **Authentication mode.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, IssuedTokenOverTransport, or SecureConversation
- **Bootstrap policy.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, or IssuedTokenOverTransport

For the Federation scenario, specify the following identity information:

- **Server certificate.** Browse for a server certificate. For more information, see "Selecting a Certificate" on page 385.
- **Expected server DNS.** the expected identity of the server in terms of its DNS. This can be **localhost** or an IP address or server name.

For the Federation scenario, specify the following STS (Security Token Service) information:

- **Issuer address.** The address of the issuer of the STS. This can be **localhost,** an IP address, or a server name.
- **Referenced file.** The file that references the binding that contacts the STS (Security Token Service)

Custom Binding

The **Custom Binding** scenario enables the highest degree of customization. Since it is based upon WCF **customBinding**, it allows you to test most WCF services, along with services on other platforms such as Java that use WS - <*spec_name*> specifications.

Use the **Custom Binding** scenario to configure a custom scenario that does not comply with any of the predefined security scenarios.

For the Custom Binding scenario, specify the following server information:

- **Transport.** HTTP, HTTPS, TCP, or NamedPipe
- **Encoding.** Text, MTOM, or WCF Binary

Specify the following security information:

- **Authentication mode.** None, AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SecureConversation, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Bootstrap policy.** For SecureConversation type authentication, specify a bootstrap policy: AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Net security.** the network security. Select None, Windows stream security, or SSL stream security. For services with HTTP transport, leave the default value, **None**. To enable SSL for HTTP, choose the HTTPS transport.

If your Web Service uses Reliable messaging, enable the option, and select **Ordered** or **Not Ordered**.

Identities

Your security settings may require you to provide identity details for either the client and server, or both of them.

An example of identity details for the client, are user name/password or an **X.509** certificate.

For identity information, provide one or more authentication details as required by the service:

Username, **Password**, **Server certificate** or **Client certificate**. For information about choosing a certificate, see "Selecting a Certificate" on page 385.

Some scenarios require you to declare the expected identity of the server in terms of its DNS, SPN, or UPN identity.

- **DNS.** Provide the name of a server or use localhost.
- **SPN.** Provide the SPN identity in the domain\machine format.

➤ **UPN.** Provide the UPN identity in the user@domain format.

After setting the basic values, you can set advanced attributes as described in "Advanced Settings" on page 387.

Selecting a Certificate

This section describes how to select a certificate.

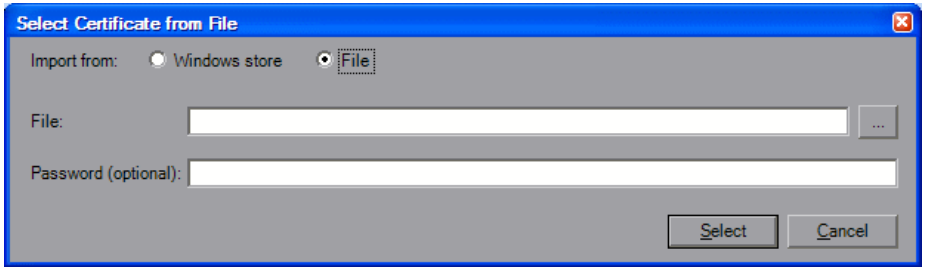
You can select a certificate from either a file or a Windows store.

Certificates Stored in Files

In this case, the certificate is stored in a file.

To select a certificate from a file:

1. Click the Browse button adjacent to the Client Certificate or Server certificate box.
2. Choose **File**. Click the Browse button to the right of the File box and locate the certificate file.



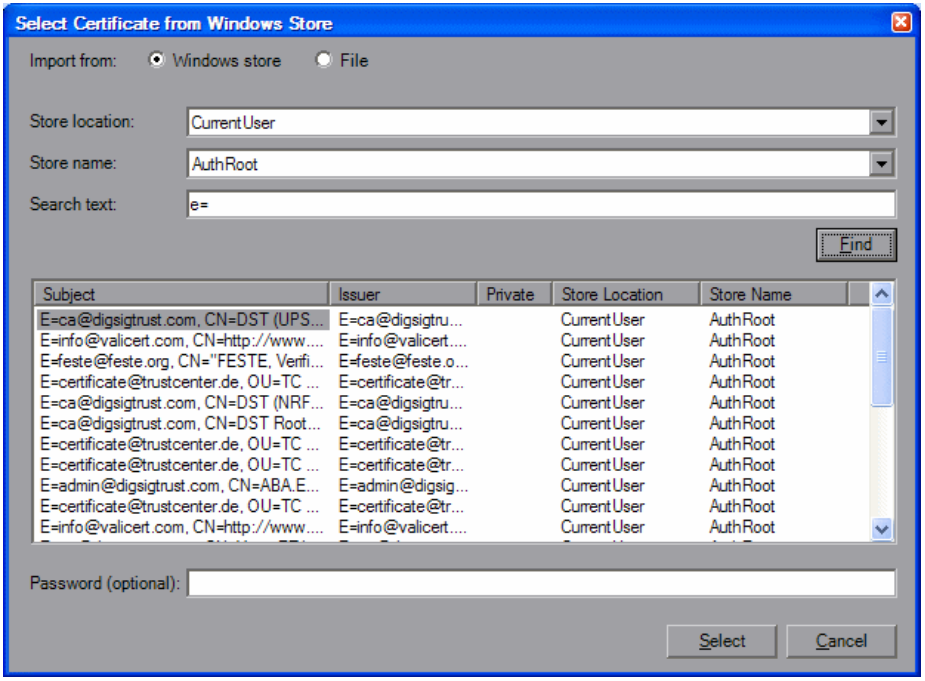
3. If required, specify a password for the private key.
4. Click **Select**. The application now references the certificate file.

Certificates From a Windows Store

In this case, the location of the certificate is in a Windows store. The dialog box provides you with a mechanism to search for the certificate.

To select a certificate from a Windows Store:

1. Click the Browse button adjacent to the Client Certificate or Server Certificate box.
2. Choose **Windows Store**. Click the Browse button to the right of the File box and locate the certificate file.
3. Select a Store location: **All**, **CurrentUser**, or **LocalMachine** (optional).
4. Select a **Store name** from the drop-down list (optional).
5. To search for all certificates, leave the **Search text** box empty. To search for a specific certificate, specify a substring of the certificate name.
6. Click **Find** to generate the list of certificates found in the store.



7. If required, specify a password for the private key.

8 Select the certificate in the list and click **Select**. The application now references the chosen certificate.

Advanced Settings

This section describes the Advanced scenario settings. Using these settings, you can customize a security scenario in the following areas: Encoding, Advanced Standards, Security, or HTTP and Proxy.

Note that not all settings are relevant for all scenarios, so some of them might be disabled or hidden depending on the scenario.

Encoding

The Encoding tab lets you indicate the type of encoding to use for the messages: **Text**, **MTOM**, or **Binary**. The default is **Text** encoding.

For each of these encoding methods, you can choose a version of WS-Addressing:

- None
- WSA 1.0
- WSA 04/08

Advanced Standards

This tab lets you configure advanced WS- standards, such as Reliable Messaging.

If your service implements the **WS-ReliableMessaging** specification, enable the **Reliable Messaging** option and set the following options:

- **Reliable messaging ordered.** indicates whether the reliable session should be ordered
- **Reliable messaging version.** WSReliableMessagingFebruary2005 or WSReliableMessaging11

Security

The Advanced security settings correspond to the **WS-Security** specifications.

For security scenarios that are based upon WCF WSHttpBinding, you can indicate the following settings:

- **Enable secure session.** Establish a security context using the WS-SecureConversation standard.
- **Negotiate service credentials.** Allow WCF proprietary negotiations to negotiate the service's security.

For **WSHttpBinding, Custom Binding**, or **WSFederationHttpBinding** WCF type scenarios, you can set the default algorithm suite and protection level:

Attribute	Meaning	Possible Values
Default Algorithm Suite	the algorithm to use for symmetric/ asymmetric encryption. These are the values from the SecurityAlgorithmSuite configuration in WCF:	➤ Basic128 ➤ Basic128Rsa15 ➤ Basic128Sha256 ➤ Basic128Sha256Rsa15 ➤ Basic192 ➤ Basic192Rsa15 ➤ Basic192Sha256 ➤ Basic192Sha256Rsa15 ➤ Basic256 ➤ Basic256Rsa15 ➤ Basic256Sha256 ➤ Basic256Sha256Rsa15 ➤ TripleDes ➤ TripleDesRsa15 ➤ TripleDesSha256 ➤ TripleDesSha256Rsa15
Protection Level	Should the SOAP Body be encrypted/signed	None, Sign, and EncryptAndSign (default)

For **Custom Binding** or **WSFederationHttpBinding** WCF type scenarios, you can customize the security settings in greater detail. The following table describes the options and their values:

Attribute	Meaning	Possible Values
Message Protection Order	The order for signing and encrypting	➤ SignBeforeEncrypt ➤ SignBeforeEncrypt-AndEncryptSignature ➤ EncryptBeforeSign
Message Security Version	The WS-Security security version	A list of the current versions
Security Header Layout	The layout for the message header	➤ Strict ➤ Lax ➤ LaxTimeStampFirst ➤ LaxTimeStampLast
Key Entropy Mode	The entropy mode for the security key.	➤ Client Entropy ➤ Security Entropy ➤ Combined Entropy

You can enable or disable the following options:

- **Require derived keys.** Indicates whether or not to require derived keys.
- **Require security context cancellation.** Disabling this option implies that stateful security tokens will be used in the **WS-SecureConversation** session (if enabled).
- **Include timestamp.** Includes a timestamp in the header.
- **Allow serialized token on reply.** Enables the reply to send a serialized token.
- **Require signature confirmation.** Instructs the server to send a signature confirmation in the response.

For X.509 certificates, you can specify values for the following items:

Attribute	Meaning	Possible Values
X509 Inclusion Mode	When to include the X509 certificate	➤ Always to Recipient ➤ Never ➤ Once ➤ AlwaysToInitiator
X509 Reference Style	How to reference the certificate	➤ Internal ➤ External
X509 require derived keys	Should X509 certificates require derived keys	➤ Enable - Yes ➤ Disable - No
X509 key identifier clause type	The type of clause used to identify the X509 key.	➤ Any ➤ Thumbprint ➤ IssuerSerial ➤ SubjectKeyIdentifier ➤ RawDataKeyIdentifier

HTTP and Proxy

This tab lets you set the HTTP and Proxy information for your test.

HTTP (S) Transport

The following table describes the HTTP(S) Transport options:

Option	Meaning	Possible Values
Transfer Mode	The transfer method for requests/responses	Buffered, Streamed, StreamedRequest, StreamedResponse
Allow Cookies	Enable cookies	Enabled/Disabled
Keep-Alive Enabled	Enable keep-alive connections	Enabled/Disabled
Authentication Scheme	HTTP authentication method	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

Option	Meaning	Possible Values
Realm	The realm of the authentication scheme	Any URL
Require Client Certificate	For SSL transport, require a certificate	Enabled/Disabled

Proxy Information

If the Web service's transport uses a proxy server, you can specify its details in the **Security** tab. The following table describes the proxy options:

Option	Meaning	Possible Values
Use Default Web Proxy	Use machine's default proxy settings	Enabled/Disabled
Bypass Proxy on Local	Ignore proxy when the service is on the local machine	Enabled/Disabled
Proxy Address	the proxy server	Any URL
Proxy Authentication Scheme	HTTP authentication method on Proxy	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

Customizing Your Security Model

The built-in scenarios with the scenario editor allow you to test most Web Services that use advanced standards.

However, you may still need to manually change a configuration file.

To modify a configuration file:

1 Select or set up a scenario as described in "Setting the Security Scenario for a Service" on page 375.

2. Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
3. Navigate to the inner folder **%Script Root%/WSDL/@config**. This folder contains one or more .stss files.
4. Open the relevant .stss file. If the directory only contains one .stss file, open it with a text editor. If it contains more then one, open the **configurationIndex.xml** to determine which .stss file matches the service you currently try to configure.
5. Modify the configuration file as described in the sections below. After you modify the file, do not update the configuration again from the Manage Services **Protocol and Security** tab, as this will override the changes.

The following are the capabilities that can be achieved by using the configuration files.

Increasing Response Buffer Capacity

In cases where you expect a large response, as is common when using MTOM, replay may issue the following error:

Error: The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

To solve this, add the **maxReceivedMessageSize** attribute to the **httpTransport** element and configure it to use a larger size. For example:

```
<protocols scenario="customBinding xmlns="http://hp/ServiceTest/config">"
    <customization>
        <mtomMessageEncoding />
                    <httpTransport maxReceivedMessageSize="6000000" />
      </customization>
</protocols>
```

Customizing Windows Credentials

Some of the security scenarios use the Windows credentials. This is common with services utilizing WCF WsHttpBinding SPNEGO and Kerberos. By default, Service Test will use the currently logged in user as the client identity. You can instruct Service Test to use the identity of another user by modifying the appropriate configuration file. If any of the elements are already present in the configuration file, update them with the new information—do not duplicate the element.

```
<protocols …>
    <identities>
        <client>
            <windowsCredentials>
                <username>myUser</username>
                <password>myPassword</password>
                <domain>myDomain</domain>
            </windowsCredentials>
        </client>
    </identities>
    ...
</protocols>
```

Using a Logical Address (clientVia Behavior)

This section describes how to specify a logical address instead of a physical one emulating **clientVia** behavior. In this case, you send a message to an intermediate service that submits it to the actual server. This may also happen when you send the message to a debugging proxy.

In such cases it may be useful to separate the physical address to which the message is actually sent, from the logical address for which the message is intended. The logical address may be the physical address of the final server or any name. It appears in the SOAP message as follows:

```
<wsa:Action>http://myLogicalAddress<wsa:Action>
```

Configuring the Logical Address

The logical address is determined in the Service Test user interface. By default, it is the address specified in the WSDL. You can override this address from the Manage Services dialog box.

Configuring the Physical Address

To set the physical address, you need to modify the configuration file. Under the **protocols** element, modify the **behaviors** element. Make sure to change the logical address to the correct one.

```
<protocols scenario="customBinding" xmlns="http://hp/ServiceTest/config">
…
<behaviors>
    <endpointBehaviors>
        <behavior>
            <clientVia viaUri="http://MyLogicalAddress" />
        </behavior>
    </endpointBehaviors>
</behaviors>
</protocols>
```

Note: When the above behavior is not present in the configuration file (as in the default behavior), then the logical address will also be the physical one, and requests will be sent to it.

Parameterizing Security Elements

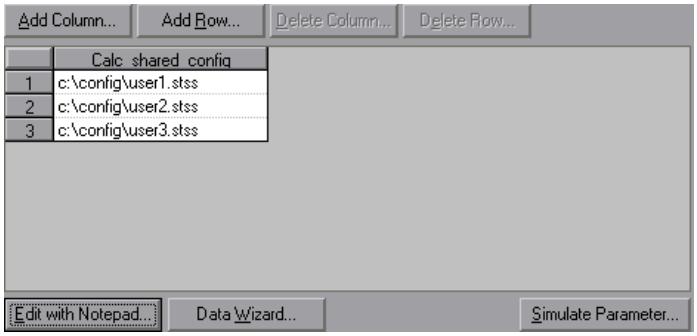
Scripts that utilize the new security model can be parameterized in the normal way. You can also parameterize the security elements independently. For example, in a username-based security scenario, you might want each Vuser or iteration to use a different user name.

To create parameters individually for each of the security elements:

1. Open the scenario editor. Select **SOA Tools** > **Security Scenario Editor**.
2. Set up and save a scenario for each Vuser. We recommend you use the names **user1**, **user2**, and so forth, and save them in a new folder, **%script root%/WSDL/referencedConfig**.
3. Open the Parameter List window. Select **Vuser** > **Parameters List**.

4 Create a new parameter, **<ServiceName>_shared_config**. Replace the <ServiceName> with the case-sensitive name of the service you are testing. To determine the exact name of the service, click **Manage Services** to see the list of services.

5 In the parameters dialog box, add the file names of the security scenarios with their .stss extensions as parameter values. You can use a relative path, which is relative to the script root folder. Click **Add Row** to add multiple values.



6 Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

7 Select **Shared Scenario**. Click the Browse button and enter the parameter name, **<ServiceName>_shared_config**, in the test box.

Simulating Users with Iterations

Many of the security scenarios establish a session with the server. For example, every scenario that uses **WS-SecureConversation** establishes a server session. This session is established when the first operation is executed and ends when the script is finished. By default, Service Test closes all sessions after each iteration and opens them again when the next iteration begins. This implies that every iteration simulates a new session and Vuser.

When working with multiple iterations, this may not be the desired effect—you may prefer to keep the original session active and not open a new session for each iteration. This applies when load testing through the LoadRunner Controller or when setting multiple iterations in the run-time settings.

You can override this behavior so that only the first iteration will establish a new session, while all subsequent ones will continue to use the open session. This simulates a user who repeatedly performs an action using the same session.

To determine which simulation mode to use, choose the one which is most appropriate to what you are simulating. For example, if you are simulating a load test where most of the actions are performed repeatedly by the same user in a single session, use the above configuration. If you are unsure, leave the default settings.

To configure your environment to use the same session for all iterations:

1. Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
2. Open **default.cfg** file in a text editor.
3. In the **[WebServices]** section, add in a row under the toolkit.

```
[WebServices]
Toolkit=.Net
SimulateNewUserInNewIteration=0
```

   If you are using the Axis toolkit or if you configured other settings, the file contents may differ.

4. Save and close the file.

Tips and Guidelines

This section provides a quick summary of using Service Test for WCF, general security, and advanced standards testing.

WCF

How do I test a WCF service?

Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF** node and choose the relevant scenario according to its binding. If you could not find an appropriate binding, choose the **customBinding** scenario since it can test any other binding.

How do I test a WCF service that uses WSHttpBinding?

WSHttpBinding is one of the most popular bindings in WCF. In order to use this binding, click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF > By client authentication type** node and choose the client credential type that you use in your binding. This value corresponds to the **MessageClientCredentialType** property of the WCF's WSHttpBinding.

Windows authentication is the default value for a new WCF service. If you are using the WCF default settings for your service, use this option. Other options are username, certificate, or none. A username can be at the message level or at the transport level (equivalent to **TransportWithMessageCredential** in WCF).

For some scenarios you should indicate whether to use the WCF proprietary negotiation mechanism to get the service credentials.

Use the Advanced scenario properties to control the usage of a secure session.

How do I test a WCF service that uses CustomBinding?

Choose a scenario type of **WCF** > **Custom Binding** as described in "Scenario Types" on page 378. You can then customize many binding elements, such as your transport method, encoding, security, and reliable messaging.

How do I test a WCF service that uses netTcp or namedPipe transport?

Choose a scenario type of **WCF** > **Custom Binding** as described in "Scenario Types" on page 378. Configure the transport to **TCP** or **NamedPipe**.

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the **web_service_set_security_saml** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**) or Chapter 20, "Web Services - Security."

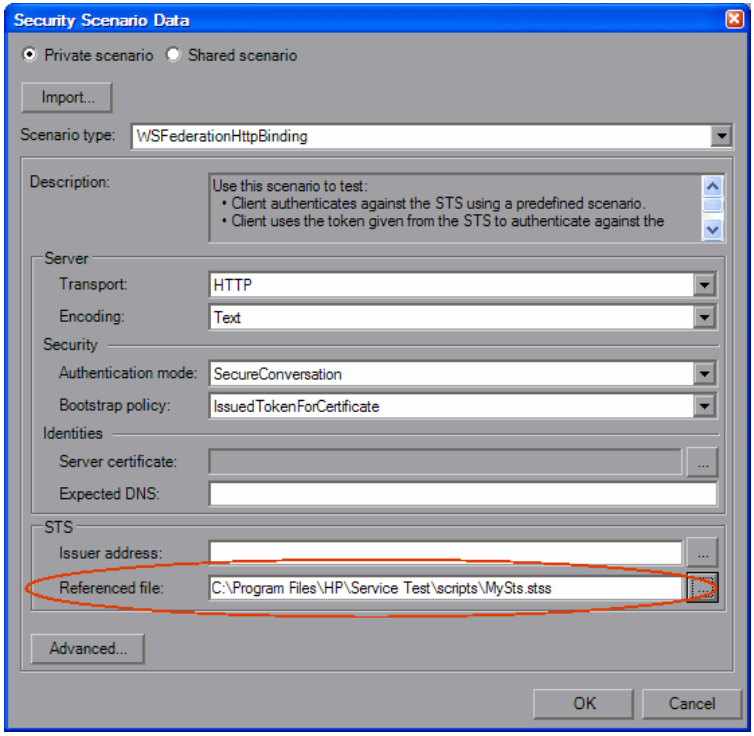
Choose the scenario **WCF** > **WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

1. Open the standalone security scenario editor. Select **SOA Tools** > **Security Scenario Editor**.
2. Click the **New** button. Configure the communication with the STS.
3. Click the **Save as** and specify a file name.

4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.



General Security

How do I test a Web Service that uses SSL?

Testing a secure site does not require any special configuration. If your service URL begins with **https**, SSL is automatically used. If in addition to SSL you are using message-level security (for example a username) then you must configure the security for the message separately using the legacy or the scenario-based model. If you use the scenario-based model, you need to configure it to use SSL by choosing an HTTPS transport or a transport credentials mode in a WSHttpBinding scenario.

How do I test a Web Service that require Windows authentication at the HTTP level?

Use the **web_set_user** function. If additional standards are required, use the Legacy security based model in conjunction with or instead of the scenario-based model.

How to I test a Web Service that uses WS-Security?

Use the scenario-based as described in this section or the legacy security using **web_service_set_security**.

How do I configure the low-level details of my WS-Security tokens?

In most cases, you can configure the low-level details as described in "Advanced Settings" on page 387. In case a very low-level control over the WS-Security tokens is required use the legacy security model. For more information, see Chapter 20, "Web Services - Security."

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the **web_service_set_security_saml** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**) or Chapter 20, "Web Services - Security."

Choose the scenario **WCF** > **WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

1. Open the standalone security scenario editor. Select **SOA Tools** > **Security Scenario Editor**.
2. Click the **New** button. Configure the communication with the STS.

3 Click the **Save as** and specify a file name.

4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.

General Protocols

How do I indicate not to use any advanced configurations?

As a scenario type, choose **<no scenario>**.

If you selected a scenario and during replay you receive errors, it is possible that you do not need an advanced scenario. Choose **<no scenario>** to cancel the existing selection and rerun the script.

How do I test a Web Service that uses MTOM?

Choose the **MTOM** scenario. If additional security is required, use one of the other scenarios. In the Advanced dialog box, set the encoding to MTOM. For more information, see "Advanced Settings" on page 387.

How do I change the WS-Addressing version of a service?

By default, the .NET toolkit uses WS-Addressing 2004/03, while the Axis toolkit does not use any addressing. To override this behavior, choose the **Plain SOAP** scenario and select the WS-Addressing version. Other supported versions are 2004/08, 1.0, and None. If your service requires additional standards, such as security, use the appropriate scenario and configure the addressing version from the **Encoding** tab in the Advanced window. For more information, see "Advanced Settings" on page 387.

22

Web Services - Transport Layers and Customizations

You can customize Web Service calls by setting the transport layer properties and by writing user handlers to define the behavior of the Web Service calls.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Testing Web Service Transport Layers

Web services can be sent over various transport layers. The transport layer is the protocol used to transport messages to and from the server.

VuGen allows you to configure the transport layer for your services. It fully supports HTTP/HTTPS and JMS (Java Message Service) transport layers.

The Service Test solution allows you to emulate both synchronous and asynchronous messaging. If you are working with HTTP/HTTPS transport, you can also use WS-Addressing.

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 420.

Configuring the Transport Layer

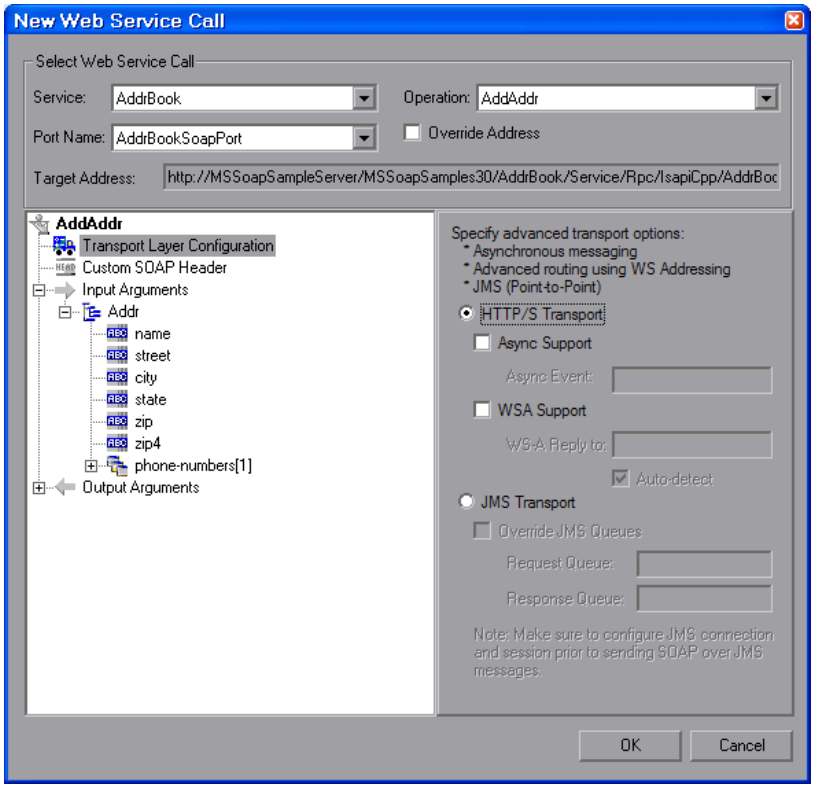
VuGen allows you to set the transport layer for your services. The transport layer indicates how to transport messages to and from the server—HTTP/HTTPS or JMS (Java Message Service).

For HTTP/HTTPS, see "Sending Messages over HTTP/HTTPS" on page 405. For more information about using the JMS transport, see "Understanding JMS" on page 406.

To choose a transport layer:

1 Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the **Transport Layer Configuration** node.



3 Choose the desired transport mode: **HTTP/S Transport** or **JMS Transport**.

For more options, see "Sending Messages over HTTP/HTTPS" below or "Using JMS as a Transport Layer" on page 407.

Sending Messages over HTTP/HTTPS

HTTP is used for sending requests from a Web client, usually a browser, to a Web server. HTTP is also used to return the Web content from the server back to the client.

HTTPS handles secure communication between a client and server. Typically, it handles credit card transactions and other sensitive data.

The typical request and response mechanism is synchronous. In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls in conjunction with WS-Addressing. For more information, see "Sending Asynchronous Calls with HTTP/HTTPS" on page 411.

Service Test also lets you emulate asynchronous messaging for your Web Service call. For more information, see "Sending Asynchronous Calls with JMS" on page 418.

Understanding JMS

JMS is a J2EE standard for sending messages, either text or Java objects, between Java clients.

There are two scenarios for communication:

Peer-to-Peer. Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.

Publish-Subscribe. Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

VuGen supports point-to-point communication by allowing you to send and receive JMS messages to and from a queue.

Before you can send messages over JMS transport, you need to configure several items that describe the transport:

- **JNDI initial context factory.** The class name of the factory class that creates an initial context which will be used to locate the JMS resources such as JMS connection factory or JMS queue.

- **JNDI provider.** The URL of the service provider which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- **JMS connection factory.** The JNDI name of the JMS connection factory.

In addition, you can set a timeout for received messages and the number of JMS connection per process.

You can configure all of these settings through the JMS run-time settings, as described in "Web Services JMS Run-Time Settings" on page 331.

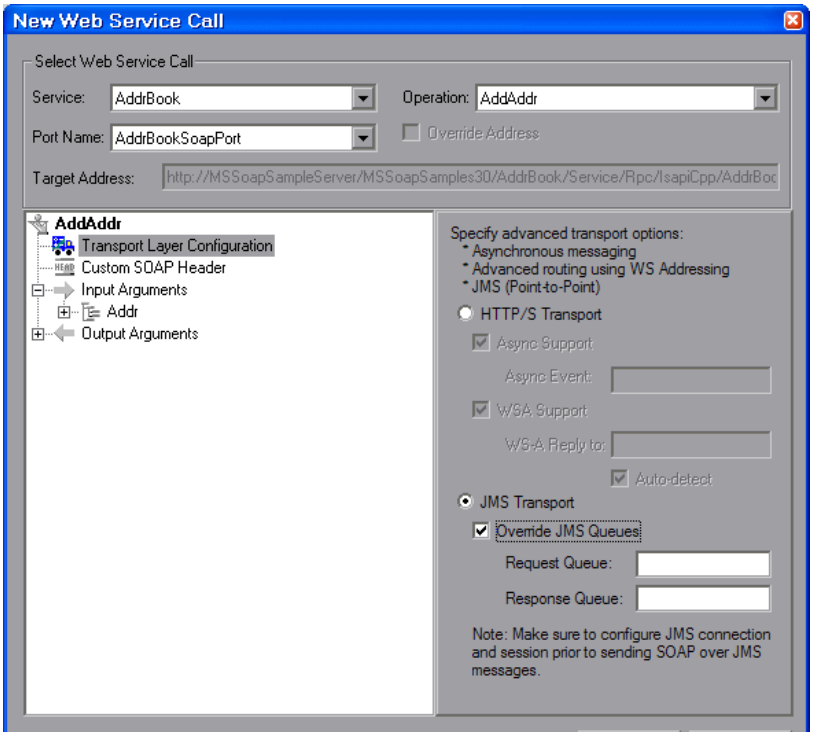
Using JMS as a Transport Layer

The JMS transport for Web services allows you to send SOAP messages using a JMS transport instead of the common HTTP transport.

To send messages using the JMS transport layer:

1. Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the **Transport Layer Configuration** node and select **JMS Transport**.



3 Configure the JMS run-time settings before running the script, as described in "Web Services JMS Run-Time Settings" on page 331.

The procedure above describes how to send synchronous JMS messages. For information on emulating asynchronous messages over JMS, see "Sending Asynchronous Calls with JMS" on page 418.

You can also send messages over JMS, even without creating Web Service calls. For example, using VuGen, you can:

- record SOAP messages using a standard Web protocol, and send them to a queue through JMS transport functions.
- manually send and receive general JMS messages from designated queues.

For more information, see "JMS Script Functions" below.

JMS Script Functions

VuGen uses its API functions to implement the JMS transport. Each function begins with a **jms** prefix:

Function Name	Description
jms_receive_message_queue	Receives a message from a queue
jms_send_message_queue	Sends a message to a queue.
jms_send_receive_message_queue	Sends a message to a specified queue and receives a message from a specified queue.
jms_set_general_property	Sets a general property in the user context.
jms_set_message_property	Sets a JMS header or property for the next message to be sent.

The JMS steps/functions are only available when manually creating scripts—you cannot record JMS messages sent between the client and server.

For additional information about the JMS functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

JMS Message Types

Each JMS message is composed of:

- **Header.** contains standard attributes (Correlation ID, Priority, Expiration date).
- **Properties.** custom attributes.
- **Body.** text or binary information.

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

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. Otherwise, it sends it 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 *Online Function Reference*.

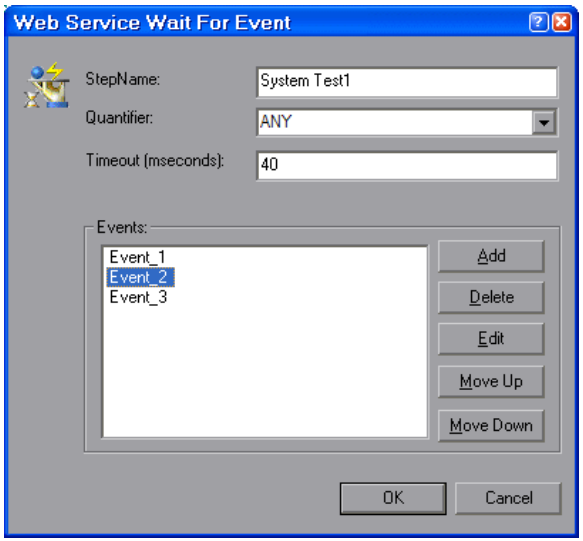
Sending Asynchronous Messages

You can use VuGen to emulate both synchronous and asynchronous messaging.

In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

Sending Asynchronous Calls with HTTP/HTTPS

This following section describes how to use asynchronous calls in HTTP/HTTPS. You use a **Wait for Event** step to instruct Vusers to wait for the response of previous asynchronous requests before continuing. The listener blocks the execution of the service until the server responds.



When adding a Web Service Wait for Event step:

- **Quantifier.** The quantifier indicates whether the Vuser should wait for **ALL** events to receive a response or **ANY,** just one of them. **ANY** returns the name of the first event to receive a response. **ALL** returns one of the event names.
- **Timeout.** the timeout in milliseconds. If no events receive responses in the specified timeout, then **web_service_wait_for_event** returns a NULL.
- **Events.** a list all of the asynchronous events for which you want to wait.

When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

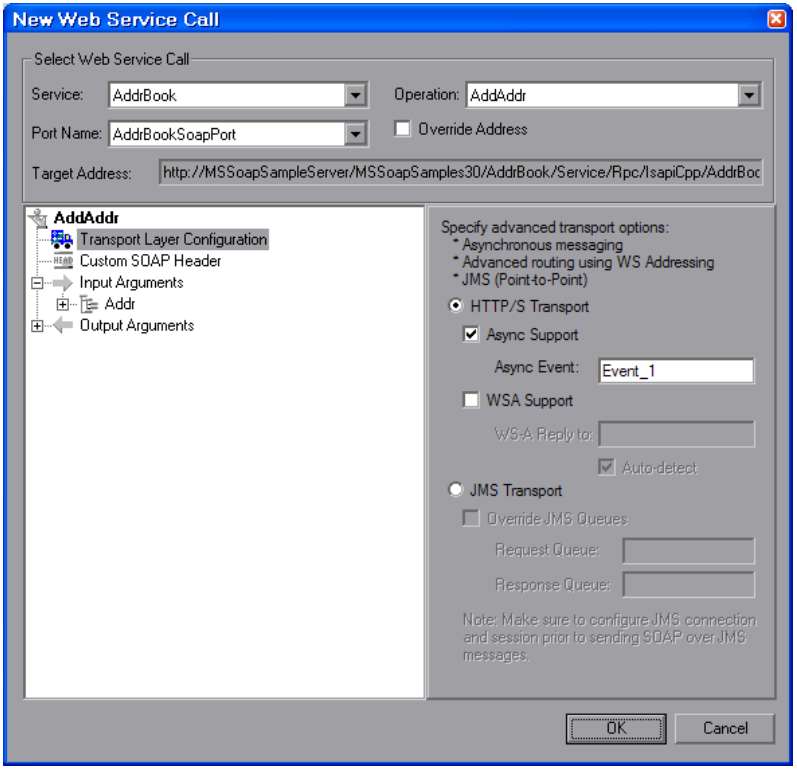
For additional information about the **Web Service Wait for Event** step or **web_service_wait_for_event** function, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

When setting up an asynchronous message, you can set the location to which the service responds when it detects an event using WS-Addressing. For more information, see "WS-Addressing" on page 414.

To send an asynchronous message using HTTP/S as a transport protocol:

1 In Tree view, select the **Step Properties** tab and select the **Transport Layer Configuration** node.

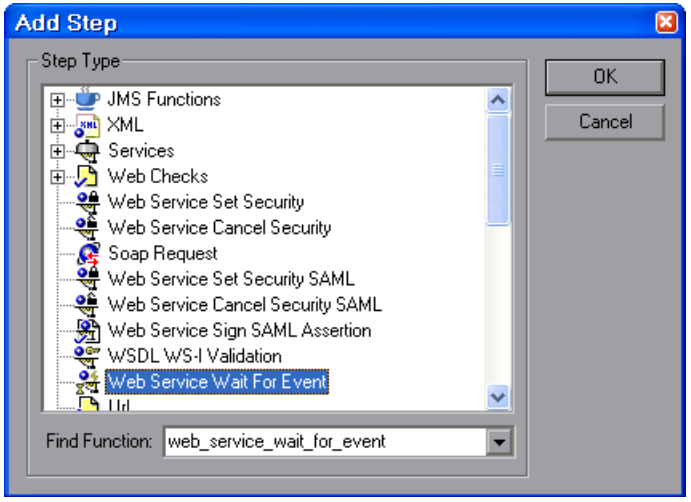
2 Choose **HTTP/S Transport** and select the **Async Support** option.



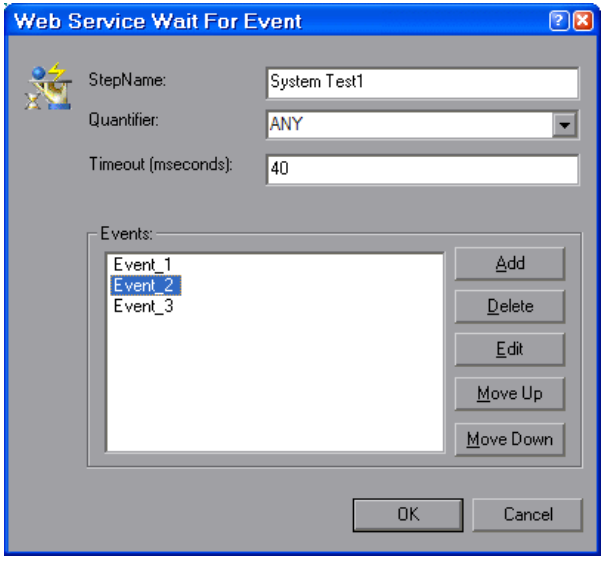
3 Type an event name in the **Async Event** box.

4 Click **OK** to generate the Web Service call.

5 Add a **Wait for Event** step. Select **Insert** > **New Step** and choose **Web Service Wait for Event**.



6 Specify a step name, a quantifier, and a timeout. Click **Add** and insert the name of the event that you defined in the previous step.



In Script view, VuGen indicates asynchronous messaging with the added parameter, **AsyncEvent**.

```
web_service_call("StepName=EchoString_101",
        "SOAPMethod=EchoRpcEncoded.EchoSoap.EchoString",
        "ResponseParam=response1",
        "Service=ExtendedECHO_rpc_encoded",
        "AsyncEvent=Event_1",
        "Snapshot=t1157371707.inf",
        BEGIN_ARGUMENTS,
        "sec=7",
        "strString=mytext",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "EchoStringResult=first_call",
        END_RESULT,
        LAST);
```

The **AsyncEvent** flag instructs the Vuser to wait for the response of previous asynchronous service requests.

For synchronous messages, create a standard Web Service call, and do not enable the **Async Support** option.

WS-Addressing

WS-Addressing is a specification that allows Web Services to communicate addressing information. It does this by identifying Web service endpoints in order to secure end-to-end endpoint identification in messages. This allows you to transmit messages through networks that have additional processing nodes such as endpoint managers, firewalls, and gateways. WS-Addressing supports Web Services messages traveling over both synchronous or asynchronous transports.

The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

An optional **WSAAction** argument allows you to define a SOAP action for instances where transport layers fails to send a message.

The following example illustrates a typical SOAP message using WS-Addressing, implemented in the background by VuGen.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2004/12/addressing">
   <S:Header>
      <wsa:MessageID>
         http://example.com/SomeUniqueMessageIdString
      </wsa:MessageID>
      <wsa:ReplyTo>
         <wsa:Address>http://myClient.example/someClientUser</wsa:Address>
      </wsa:ReplyTo>
<wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
      </wsa:FaultTo>
      <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
      <wsa:Action>http://myserver.example/DoAction</wsa:Action>
   </S:Header>
   <S:Body>
      <!-- Body of SOAP request message -->
   </S:Body>
</S:Envelope>
```

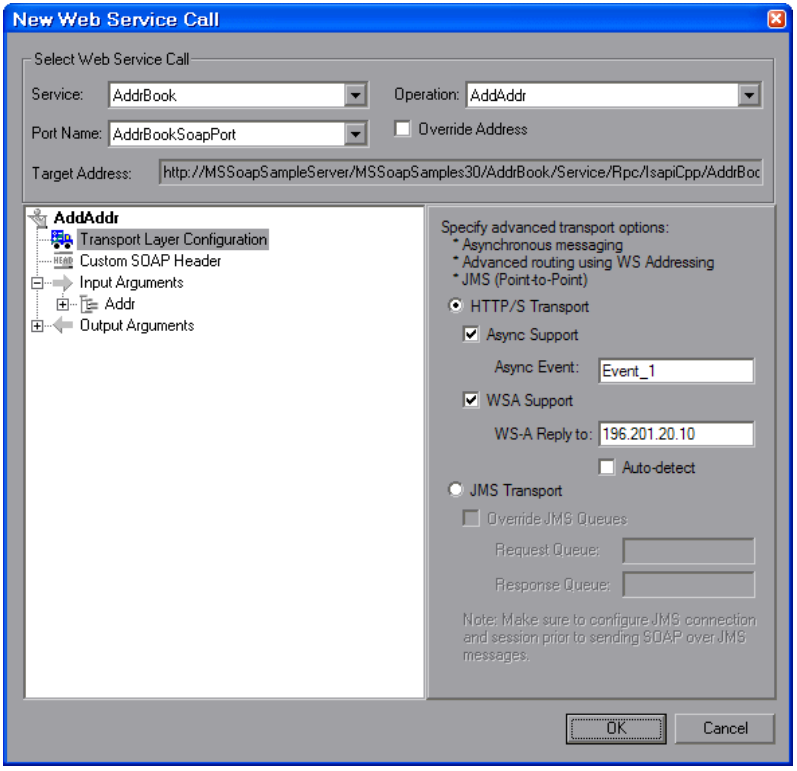
In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
web_service_call("StepName=Add_101",
         "SOAPMethod=Calc.CalcSoap.Add",
         "ResponseParam=response",
         "AsyncEvent=Event_1",
            "WSAReplyTo=212.199.95.138",
         "WSDL=http://lab1/WebServices/CalcWS/Calc.asmx?wsdl",
         "UseWSDLCopy=1",
         "Snapshot=t1153825715.inf",
         BEGIN_ARGUMENTS,
            "first=1",
            "second=2",
         END_ARGUMENTS,
         BEGIN_RESULT,
            "AddResult=Param_AddResult1",
         END_RESULT,
         LAST);
```

You can issue WS-Addressing calls in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, leave the **Async Event** box empty in the Transport Layer options. In Script view, remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

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:

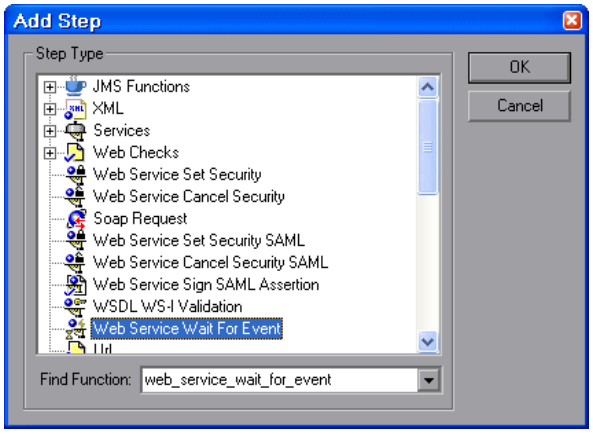
➤ Select the **Async Support** option. This is only enabled for HTTP/S type transport.

➤ Provide an event name in the **Async Event** box. This can be an arbitrary name.

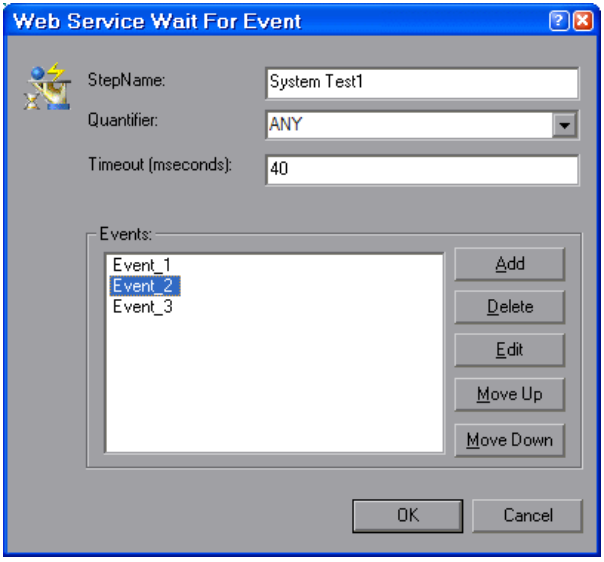
3 Select **WSA Support**. In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. Autodetect is useful when running the same script on several different machines. The server will reply to the specified location when the event occurs.

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

5 Instruct the Vuser to wait for an event. Select **Insert** > **New Step** and add a **Web Service Wait For Event** step after the Web Service call step.



6 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.



7 Use the **Edit**, **Move Up**, and **Move Down** buttons to manipulate the events.

8 Click **OK**.

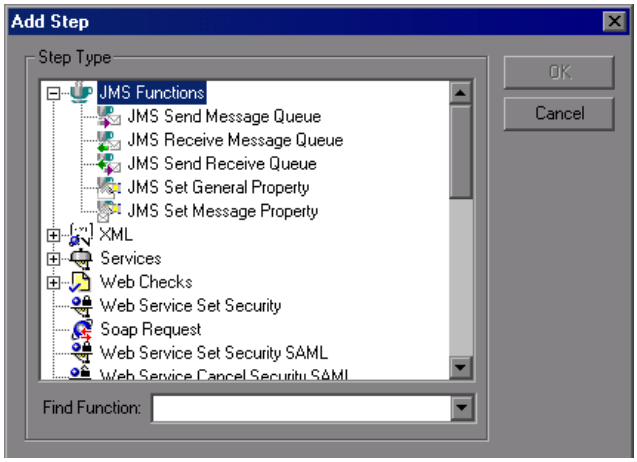
Sending Asynchronous Calls with JMS

This section describes how to send asynchronous messages using JMS.

To implement this, you send the request or retrieve the response using JMS steps—not Web Service calls. **JMS Send Message Queue** sends a message to a queue. **JMS Receive Message Queue** receives a message from the queue.

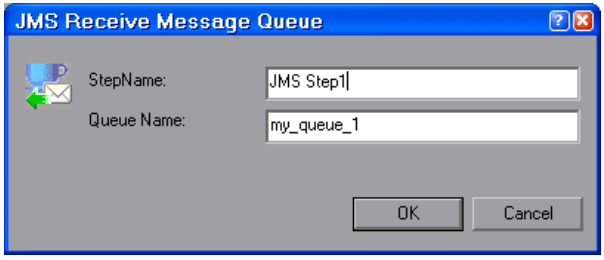
To send and receive asynchronous messages using JMS:

1 Click within the script at the desired location. Select **Insert** > **New Step** and expand the **JMS Functions** node.



2 Select a JMS function to set up the message queue information: **Send Message Queue** or **JMS Receive Message Queue**.

3 Click **OK** to open the properties the JMS functions.



4 Specify a queue name and click **OK** to generate the JMS functions.

For JMS synchronous messages, create a standard Web Service call, select JMS transport, and if desired, specify the queue information.

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

VuGen provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are:

User Handlers

User Handlers are open API through which you can perform the following operations:

- ➤ Get and set the request/response SOAP envelopes
- ➤ Override the transport layer
- ➤ Get and set the request/response content type
- ➤ Get and Set values for LoadRunner parameters
- ➤ Retrieve a configuration argument from the script
- ➤ Issue messages to the execution log
- ➤ Fail an execution

For information on setting up the user handlers, see "Setting up the User Handlers" below.

For sample user handlers, see "User Handler Examples" on page 426.

Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration. For more information, see "Using Custom Configuration Files" on page 430.

Setting up the User Handlers

You can set up a user handler directly in a script, or you can implement through a DLL. You can apply a handler locally or globally. For details, refer to the following sections:

- ❑ Defining a Handler Function in a Script
- ❑ Creating a Custom User Handler as a DLL
- ❑ Configuring the User Handler

❑ Implementing the User Handler

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)
{
…
}
```

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);
```

VuGen recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_SUCCEEDED_AND_MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in **SoapEnvelopeParam**.

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    return 0;
```

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 **LoadRunner/include** folder, contains many in-line comments and descriptions.

VuGen 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 **LoadRunner/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 **LoadRunner/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 DLL user handler 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: Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the **LoadRunner/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 **LoadRunner/include** folder.

User Handler Examples

The following section describes how to create user handlers for several common issues:

- .NET Filters
- Overriding the Transport Layer
- Including MIME Attachments

.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
```

Overriding the Transport Layer

You can write a user handler function to override the transport layer. In this case, VuGen 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);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the **web_service_call**:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
  "UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
   "<cls>"
    "<i>123456789</i>"
    "<s>abcde</s>"
   "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
   LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the file you want to send and its content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the **web_service_call**:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same **web_service_call**, add the following code:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;
FileName=C:\\temp\\results.discomap",
```

Using Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration.

The standard .NET configuration file, **mmdrv.exe.config**, is located in the **LoadRunner/bin** folder.

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.

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.

In addition, the configuration file contains security information. You can configure whether or not to allow unsigned test certificates.

By default, VuGen 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”
```

23

Web Services - User Handlers and Customization

Service Test allows you to customize your script with user handlers and configuration files.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Customizing Web Service Script Behavior

VuGen provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are user handlers and configuration files.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see below.

Configuration files let you customize advanced settings such as security information and the WSE configuration. For more information, see "Using Custom Configuration Files" on page 443.

Setting up User Handlers

User Handlers are open API through which you can perform the following operations:

- ➤ Get and set the request/response SOAP envelopes
- ➤ Override the transport layer
- ➤ Get and set the request/response content type
- ➤ Get and Set values for LoadRunner parameters
- ➤ Retrieve a configuration argument from the script
- ➤ Issue messages to the execution log
- ➤ Fail an execution

You can set up a user handler directly in a script, or you can implement through a DLL. You can apply a handler locally or globally. For details, see the following sections:

- ❑ Defining a Handler Function in a Script
- ❑ Creating a Custom User Handler as a DLL
- ❑ Configuring the User Handler
- ❑ Implementing the User Handler

For sample user handlers, see "User Handler Examples" on page 439.

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)
{
…
}
```

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);
```

VuGen recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_SUCCEEDED_AND_MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in **SoapEnvelopeParam**.

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    return 0;
```

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 **LoadRunner/include** folder, contains many in-line comments and descriptions.

VuGen 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 **LoadRunner/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 **LoadRunner/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 DLL user handler 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: Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the **LoadRunner/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 **LoadRunner/include** folder.

User Handler Examples

The following section describes how to create user handlers for several common issues:

- .NET Filters
- Overriding the Transport Layer
- Including MIME Attachments

.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
```

Overriding the Transport Layer

You can write a user handler function to override the transport layer. In this case, VuGen 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);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the **web_service_call**:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
  "UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
   "<cls>"
    "<i>123456789</i>"
    "<s>abcde</s>"
   "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
   LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the file you want to send and its content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the **web_service_call**:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same **web_service_call**, add the following code:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;
FileName=C:\\temp\\results.discomap",
```

Using Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration.

The standard .NET configuration file, **mmdrv.exe.config**, is located in the **LoadRunner/bin** folder.

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.

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.

In addition, the configuration file contains security information. You can configure whether or not to allow unsigned test certificates.

By default, VuGen 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”
```

24

Web Services - Negative Testing

Using VuGen, you can test your Web Service using different testing methodologies, such as positive or negative testing.

This chapter includes:

The following information only applies to Web Services/SOA Vuser scripts.

About Applying Testing Methodologies

When performing a functional test for your Web Service, you should approach the testing in a variety of ways. The most common type of testing is called **Positive Testing**—checking that the service does what it was designed to do.

In addition, you should perform **Negative Testing**, to confirm that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued an appropriate error—a SOAP Fault.

To illustrate this, consider a form accepting input data—you apply positive testing to check that your Web Service has properly accepted the name and other input data. You apply negative testing to make sure that the application detects an invalid character, for example a letter character in a telephone number.

When your service send requests to the server, the server responds in one of the following ways:

- **SOAP Result.** A SOAP response to the request
- **SOAP Fault.** A response indicating that the SOAP request was invalid. Negative Testing applies only to SOAP faults.
- **HTTP Error.** An HTTP error, such as Page Not Found, unrelated to Web Services.

VuGencan check for a standard SOAP result or a SOAP fault responses—it always fails on HTTP errors. For example, if your Web Service attempts to access a Web page that cannot be found, resulting in a 404 HTTP error, the replay will fail, even if the SOAP is valid.

Understanding the Testing Settings

When creating a Web Service Call or SOAP Request in VuGen, you can indicate the type of testing you want to perform during replay:

Type of Testing	Description
Positive Testing	Accept SOAP result responses and fail on SOAP faults.
Negative Testing	Accept SOAP faults and fail on SOAP result responses.
Any Type	Accept both SOAP result and SOAP fault responses.

By default, VuGen, only performs positive testing and passes a test when it receives a SOAP result response. You can instruct VuGento perform only negative testing, or to accept any SOAP response. If you enable negative testing only, and the server issues a regular SOAP result response, the step will have a **Failed** status.

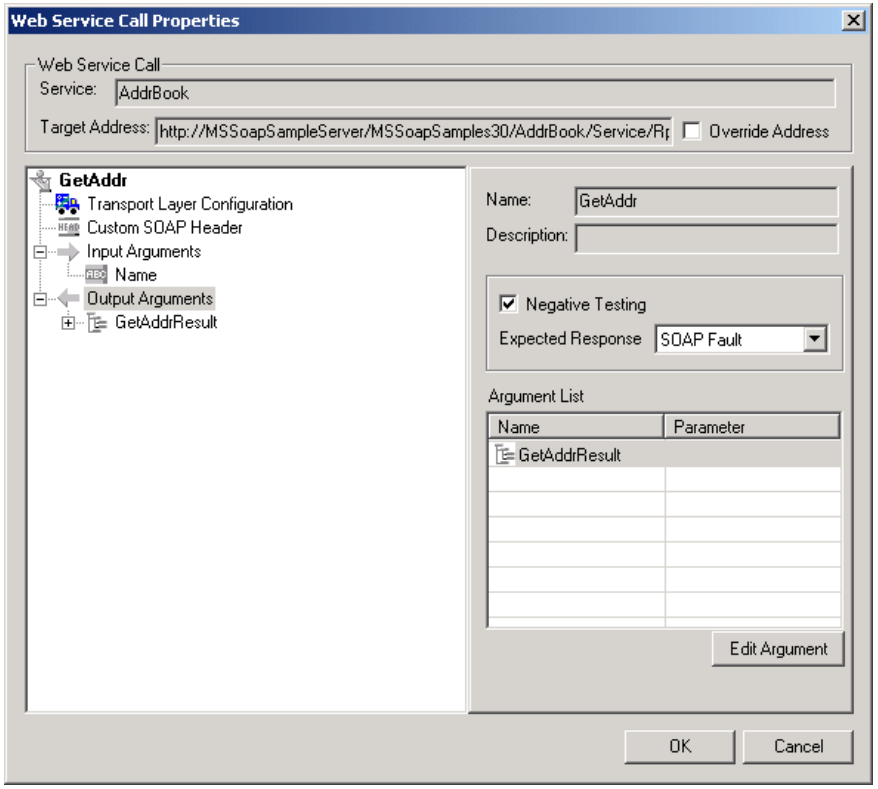
You can instruct VuGento accept any SOAP response—a SOAP result or SOAP fault. This can be useful in testing environments where you only need to send the request, using a separate function to check the SOAP at a later time. In this testing mode, steps with either a SOAP result or SOAP fault will be issued a **Passed** status.

You can check the status of the replay by viewing the Replay log and Test Results report. A failed step will be marked in red as **Failed** in the Test Results.

If you are working with Quality Center or HP Service Test Manager, the application will list the test's status based on the Expected Response setting.

Defining a Testing Method

Before replaying the script, you indicate the testing method in the Web Service Call properties dialog box—positive or negative testing, or any SOAP.



To select a testing method:

1. Select the step whose response you want to test. Open the Properties from the right-click menu.
2. Select the **Output Argument** node.
3. Choose an Expected Response.
   - To perform positive testing only, clear the **Negative Testing** check box.
   - To perform negative testing only, select the **Negative Testing** check box and choose **SOAP Fault** as the **Expected Response**.
   - To accept any type of SOAP response, select the **Negative Testing** check box and choose **Any SOAP** as the **Expected Response**.

In Script view, VuGenrepresents the testing method with the **ExpectedResponse** argument. The possible values are **SoapResult**, **SoapFault**, or **AnySoap**. In the following example, the script performs negative testing, indicated by the **SoapFault** value:

```
web_service_call( "StepName=AddAddr_101",
        "SOAPMethod=AddrBook|AddrBookSoapPort|AddAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapFault",
        "Snapshot=t1189409011.inf",
        BEGIN_ARGUMENTS,
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
```

Evaluating the SOAP Fault Value

When you replay a script that results in a SOAP fault, VuGensaves the fault to a parameter called **response**. To check the value of the SOAP fault that was returned, you evaluate the **response** output parameter using **lr_xml_find.**

In the following example, **lr_xml_find** checks for a **VersionMismatch** SOAP fault and issues an output message.

```
lr_xml_find("XML={response}",
                "FastQuery=/Envelope/Body/Fault/faultString ",
                "Value=VersionMismatch",
                LAST);

if (soap_fault_cnt >0)
    lr_output_message{"A Version Mismatch SOAP Fault occurred")
```

For more information about **lr_xml_find**, see the *Online Function Reference*. (**Help** > **Function Reference**)

25

Java Protocols - Recording

VuGen allows you to record applications or applets written in Java, in protocols such as CORBA, RMI, EJB, JMS or Jacada. You can also use VuGen's navigation tool to add any method to your script.

This chapter includes:

About Recording Java Language Vuser Scripts

Using VuGen, you can record a Java application or applet. VuGen creates a pure Java script enhanced with Vuser API Java-specific functions. After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it. Once you verify that the script is functional, you incorporate it into a LoadRunner scenario or Business Process Monitor profile.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. See Chapter 38, "Programming Java Scripts" for important information about function syntax and system configuration.

Before recording a CORBA session, verify that your application or applet functions properly on the recording machine.

Make sure that you have properly installed a JDK version from Sun on the machine running VuGen—JRE alone is insufficient. You must complete this installation before recording a script. Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions.

Note: When you load an applet or application from VuGen during recording, it may take several seconds longer than if you were to load it independent of VuGen.

VuGen provides a tool that enables you to convert a Vuser script created for Web, into Java. For more information, see "Converting Web Vuser Scripts into Java" on page 780.

After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it.

You integrate finished scripts into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Getting Started with Java Vuser Scripts

The following procedure outlines how to record Java language Vuser scripts.

1 Make sure that the recording machine is properly configured.

Make sure that your machine is configured properly for Java before you begin recording. For more information, see Chapter 38, "Programming Java Scripts" and the Readme file.

2 Create a new Vuser script.

Select The Java Record/Replay Vuser type.

3 Specify a Java protocol.

Select a protocol from the recording options.

4 Set the recording parameters and options for the script.

You specify the parameters for your applet or application such as working directory and paths. You can also set JVM, serialization, correlation, recorder, and debug recording options. See Chapter 75, "Java Recording Options"for more information.

5 Record typical user actions.

Begin recording a script. Perform typical actions within your applet or application. VuGen records your actions and generates a Vuser script.

6 Enhance the Vuser script.

Add Vuser API specific functions to enhance the Vuser script. For details, see Chapter 38, "Programming Java Scripts." You can use the built-in Java function Navigator. For more information, see "Viewing the Java Methods" on page 470.

7 Parameterize the Vuser script.

Replace recorded constants with parameters. You can parameterize complete strings or parts of a string. Note that you can define more than one parameter for functions with multiple arguments. For details, see Chapter 70, "Working with VuGen Parameters."

8 Configure the run-time setting for the script.

Configure run-time settings for the Vuser script. The run-time settings define the run-time aspects of the script execution. For the specific run-time settings for Java, see "Java and EJB Run-Time Settings" on page 1325.

9 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

For detailed information on the recording procedure, see the specific chapter for your Vuser type.

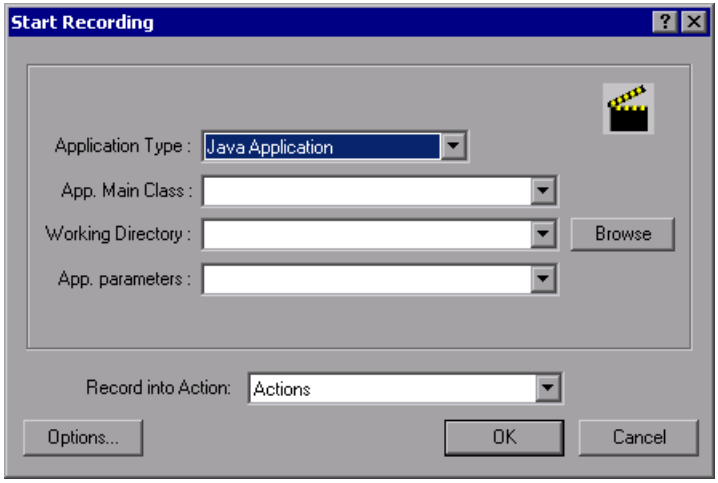
Recording Java Events

Make sure that you have properly installed a JDK version from Sun on the machine running the Vusers—JRE alone is insufficient.Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions. Before you replay a Vuser script, verify that your environment is configured properly for the JDK and relevant Java classes.

This is the general procedure for recording a Java session.

To begin recording:

1 Select **File** > **New** and select **JAVA Record Replay** from the **Java** category. The Start Recording dialog box opens.



2 In the **Application Type** box, select the appropriate value.

- **Java Applet** to record a Java applet through Sun's appletviewer.
- **Java Application** to record a Java application.
- **Netscape** or **IExplore** to record an applet within a browser.
- **Executable/Batch** to record an applet or application that is launched from within a batch file or the name of an executable file.

➤ **Listener** to instruct VuGen to wait for the batch file that initializes the configuration and runs an application before recording. This mode requires you to define the system variable _JAVA_OPTIONS as -- **Xrunjdkhook** using jdk1.2.x and higher. (For JDK 1.1.x, define the environment variable **_classload_hook=JDKhook**. For JDK 1.6 set _JAVA_OPTIONS as **-agentlib:jdhook**.)

3 Specify additional parameters according for the following chart:

Application Type	Fields to Set
Java Applet	Applet Path, Working Directory
Java Application	App. Main Class, Working Directory, App. parameters
IExplore	IExplore Path, URL
Netscape	Netscape Path, URL
Executable/Batch	Executable/Batch, Working Directory
Listener	N/A

Note: A Working Directory is necessary only if your application must know the location of the working directory (for example, reading property files or writing log files).

4 Click **Options** to open the Recording Options dialog box. You select a Java protocol: CORBA, RMI, JMS, or Jacada and set other recording properties**.** See Chapter 75, "Java Recording Options"for more information.

5 In the Record into Action box, select the section corresponding to the method into which you want to record. The Actions class contains three methods: init, action, and end, corresponding to the **vuser_init**, **Actions**, and **vuser_end** sections. The following table shows what to include into each method, and when each method is executed.

method within Actions class	Record into action	Used to emulate...	Executed during...
init	**vuser_init**	a login to a server	Initialization

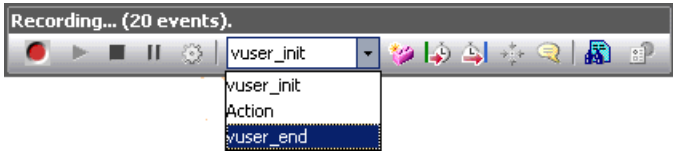
method within Actions class	Record into action	Used to emulate...	Executed during...
action	**Actions**	client activity	Running
end	**vuser_end**	a log off procedure	Finish or Stopped

Note: Make sure to import the **org.omg.CORBA.ORB** function in the **vuser_init** section, so that it will not be repeated for each iteration.

6 Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.

7 Perform typical actions within your application. Use the floating toolbar to switch methods during recording.

8 After recording the typical user actions, select the **vuser_end** method from the floating toolbar.



Perform the log off procedure. VuGen records the procedure into the **vuser_end** method of the script.

9 Click **Stop Recording** on the Recording toolbar. The VuGen script editor displays all the recorded statements.

Click **Save** and provide a name for the script.

Recording CORBA

For recording a CORBA session, you need to set the following options in the Recording Options:

- JNDI
- Use DLL hooking to attach VuGen support

Using CORBA Application Vendor Classes

Running CORBA applications with JDK1.2 or later, might load the JDK internal CORBA classes instead of the specific vendor CORBA classes. To force the virtual machine to use the vendor classes, specify the following java.exe command-line parameters:

Visigenic 3.4
-Dorg.omg.CORBA.ORBClass=com.visigenic.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.visigenic.vbroker.orb.
ORBSingleton

Visigenic 4.0
-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton

OrbixWeb 3.x
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.
singletonORB

OrbixWeb 2000
-Dorg.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.
ORBSingleton

Recording RMI over IIOP

The **Internet Inter-ORB Protocol** (IIOP) technology was developed to allow implementation of CORBA solutions over the World Wide Web. IIOP lets browsers and servers exchange complex objects such as arrays, unlike HTTP, which only supports transmission of text.

RMI over IIOP technology makes it possible for a single client to access services which were only accessible from either RMI or CORBA clients in the past. This technology is a hybrid of the JRMP protocol used with RMI and IIOP used with CORBA. **RMI over IIOP** allows CORBA clients to access new technologies such as **Enterprise Java Beans** (EJB) among other J2EE standards.

VuGen provides full support for recording and replaying Vusers using the **RMI over IIOP** protocol. Depending on what you are recording, you can utilize VuGen's RMI recorder to create a script that will optimally emulate a real user:

- **Pure RMI client.** recording a client that uses native JRMP protocol for remote invocations
- **RMI over IIOP client.** recording a client application that was compiled using the IIOP protocol instead of JRMP (for compatibility with CORBA servers).

Recording RMI

Before recording an RMI session, verify that your application or applet functions properly on the recording machine.

Before you record, verify that your environment is configured properly. Make sure that the required classes are in the classpath and that you have a full installation of JDK. For more information on the required environment settings, see Chapter 38, "Programming Java Scripts."

Recording a Jacada Vuser

The Jacada Interface Server provides an interface layer for mainframe applications. This layer separates the user interface from the application logic in order to insulate the organization from changes in standards and technologies. Instead of working with green-screen applications, the Jacada server converts the environment to a user friendly interface.

VuGen records Jacada's Java thin-client. To record communication with the Jacada server through the HTML thin-client, use the Web HTTP/HTML type Vuser. For more information, see Chapter 48, "Web (HTTP/HTML, Click and Script) Protocols."

Before replay, you must also download the **clbase.jar** file from the Jacada server. All classes used by the Java Vuser must be in the classpath—either set in the machine's CLASSPATH environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

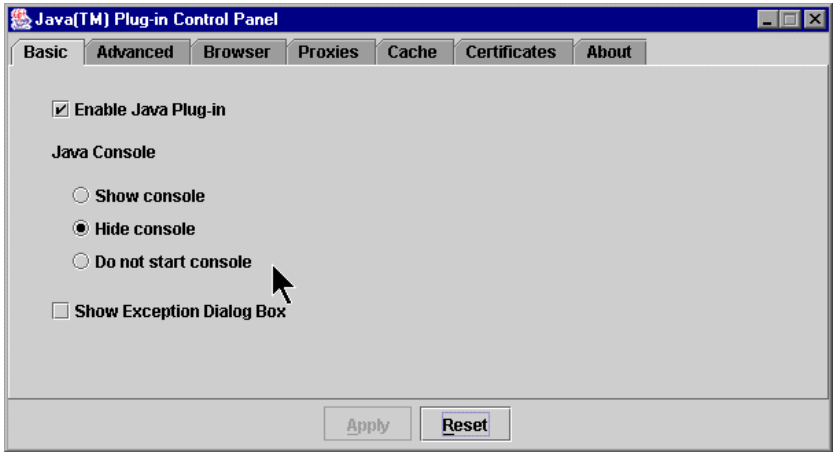
During replay, the Jacada server may return screens from the legacy system, in a different order than they appear in the recorded script. This may cause an exception in the replay. For information on how to handle these exceptions, please contact support.

Recording on Windows XP and Windows 2000 Servers

When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

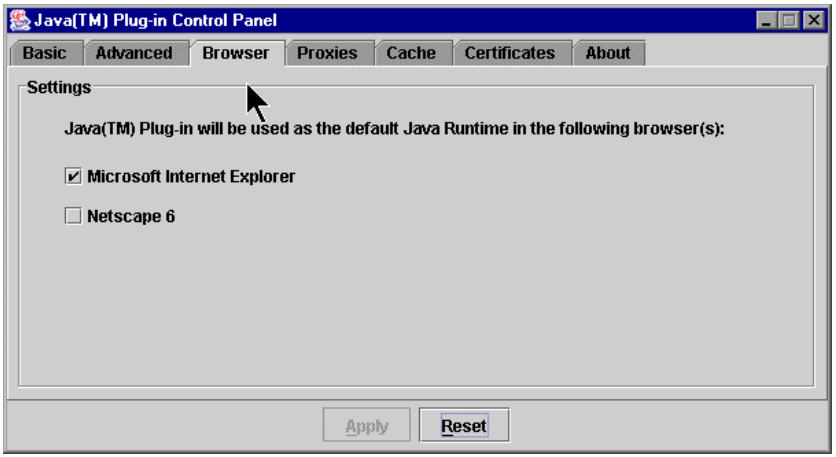
To configure your machine for recording CORBA or RMI sessions:

1 Open the Java Plug-in from the Control Panel. Select **Start** > **Settings** > **Control Panel** and open the **Java Plug-in** component. The Basic tab opens.



2 Clear the **Enable Java Plug-In** check box and click **Apply**. Then, reselect the **Enable Java Plug-In** check box and click **Apply.**

3 Open the Browser tab.



4 Clear the **Microsoft Internet Explorer** check box and click **Apply**. Then, reselect the **Microsoft Internet Explorer** check box and click **Apply.**

26

Java - Managing Vuser Scripts

VuGen allows you to record applications or applets written in Java. You can run the recorded script or enhance it using standard Java library functions and Vuser API Java-specific functions.

This chapter includes:

Understanding Java Vuser Scripts

When you record a session, VuGen logs all calls to the server and generates a script with functions. These functions describe all of your actions within the application or applet. The script also contains supplementary code required for proper playback, such as property settings, and naming service initialization (JNDI).

The recorded script is comprised of three sections:

- Imports

- Code
- Variables

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script. The **Code** section contains the Actions class and the recorded code within the **init**, **actions**, and **end** methods. The **Variables** section, after the **end** method, contains all the type declarations for the variables used in the code.

After you finish recording, you can modify the functions in your script, or add additional Java or LoadRunner functions to enhance the script. Note that if you intend to run Java Vusers as threads, the Java code you add to your script must be thread-safe. For details about function syntax, see the *Online Function Reference* (**Help > Function Reference**). In addition, you can modify your script to enable it to run as part of another package. For more information, see "Compiling and Running a Script as Part of a Package" on page 660.

Working with CORBA

CORBA-specific scripts usually have a well-defined pattern. The first section contains the ORB initialization and configuration. The next section indicates the location of the CORBA objects. The following section consists of the server invocations on the CORBA objects. The final section includes a shutdown procedure which closes the ORB. Note that pattern is not mandatory and that each one of these sections may appear multiple times within a script.

In the following segment, the script initializes an ORB instance and performs a bind operation to obtain a CORBA object. Note how VuGen imports all of the necessary classes.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;

public class Actions {

  // Public function: init
  public int init() throws Throwable {

  // Initialize Orb instance...
  MApplet mapplet = new MApplet("http://chaos/classes/", null);
  orb = org.omg.CORBA.ORB.init(mapplet, null);

  // Bind to server...
  grid = grid_dsi.gridHelper.bind("gridDSI", "chaos");
  return lr.PASS;
}
```

The org.omg.CORBA.ORB function makes the connection to ORB. Therefore, it should only be called once. When running multiple iterations, place this function in the **init** section.

In the following section, VuGen recorded the actions performed upon a grid CORBA object.

```
// Public function: action
public int action() throws Throwable {

   grid.width();
   grid.height();
   grid.set(2, 4, 10);
   grid.get(2, 4);

   return lr.PASS;
}
```

At the end of the session, VuGen recorded the shutdown of the ORB. The variables used throughout the entire recorded code appear after the **end** method and before the Actions class closing curly bracket.

```
// Public function: end
    public int end() throws Throwable {

        if (lr.get_vuser_id() == -1)
            orb.shutdown();

        return lr.PASS;
}

// Variable section
    org.omg.CORBA.ORB orb;
    grid_dsi.grid grid;
}
```

Note that the ORB shutdown statement was customized for this product. This customization prevents a single Vuser's shutdown from shutting down all other Vusers.

Working with RMI

This section describes the elements of the Java Vuser script that are specific to RMI. RMI does not have constructs (as in CORBA)—instead it uses Serializable Java objects. The first section performs a Naming Registry initialization and configuration. The next section is generated when Java objects (both Remote and Serializable) are located and casted. The following section consists of the server invocations on the Java objects. In RMI there is no specific shutdown section (unlike CORBA). Note that objects might appear multiple times within the script.

The following segment locates a naming registry. This is followed by a lookup operation to obtain a specific Java object. Once you obtain the object, you can work with it and perform invocations such as **set_sum**, **increment**, and **get_sum**. The following segment also shows how VuGen imports all of the necessary RMI classes.

```
Import java.rmi.*;
Import java.rmi.registry.*;

:
:

// Public function: action
public int action() throws Throwable {

    _registry = LocateRegistry.getRegistry("localhost",1099);

    counter = (Counter)_registry.lookup("Counter1");

    counter.set_sum(0);
    counter.increment();
    counter.increment();
    counter.get_sum();

    return lr.PASS;
    }
:
```

When recording RMI Java, your script may contain several calls to **lr.deserialize**, which deserializes all of the relevant objects. The **lr.deserialize** calls are generated because the object being passed to the next invocation could not be correlated to a return value from any of the previous calls. VuGen therefore records its state and uses **lr.deserialize** call to represent these values during replay. The deserialization is done before VuGen passes the objects as parameters to invocations. For more information, see "Using the Serialization Mechanism" on page 492.

Working with Jacada

The Actions method of a Java Vuser script using Jacada, has two main parts: properties and body. The properties section gets the server properties. VuGen then sets the system properties and connects to the Jacada server.

```
// Set system properties...
      _properties = new Properties(System.getProperties());
      _properties.put("com.ms.applet.enable.logging", "true");
      System.setProperties(_properties);

      _jacadavirtualuser = new cst.client.manager.JacadaVirtualUser();

      lr.think_time(4);
      _jacadavirtualuser.connectUsingPorts("localhost", 1100, "LOADTEST", "", "", "");
     …
```

The body of the script contains the user actions along with the exception handling blocks for the checkFieldValue and checkTableCell methods.

```
      l…
/*
try {
       _jacadavirtualuser.checkFieldValue(23, "S44452BA");
       }catch(java.lang.Exception e) {
        lr.log_message(e.getMessage());
       }
*/    l…
/*
try {
     _jacadavirtualuser.checkTableCell(41, 0, 0, "");
       }catch(java.lang.Exception e) {
        lr.log_message(e.getMessage());
       }
*/    l…
```

The **checkField** method has two arguments: field ID number and expected value. The **checkTableCell** method has four arguments: table ID, row, column, and expected value. If there is a mismatch between the expected value and the received value, an exception is generated.

By default, the try-catch wrapper blocks are commented out. To use them in your script, remove the comment markers.

In addition to the recorded script, you can add any of the Java Vuser API functions. For a list of these functions and information on how to add them to your script, see Chapter 38, "Programming Java Scripts."

Running a Script as Part of a Package

This section is not relevant for Jacada type scripts.

When creating or recording a Java script, you may need to use methods from classes in which the method or class is protected. When attempting to compile such a script, you receive compilation errors indicating that the methods are not accessible.

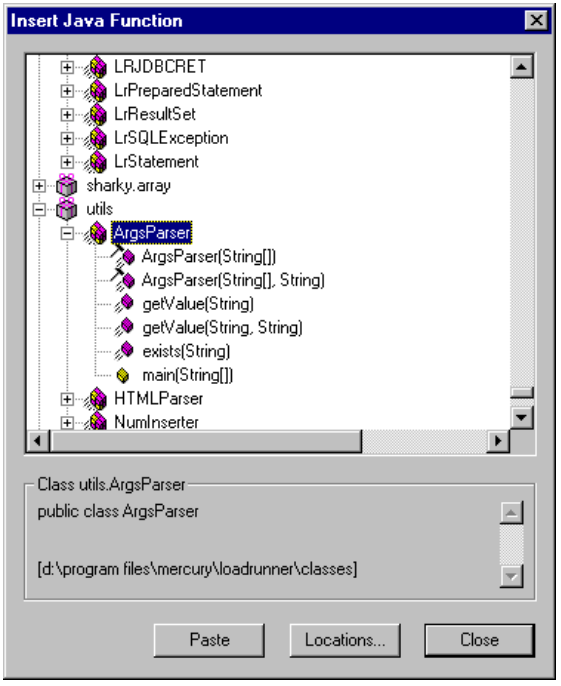
To use the protected methods, add the Vuser to the package of required methods. At the beginning of your script, add the following line:

```
package a.b.c;
```

where **a.b.c** represents a directory hierarchy. VuGen creates the a/b/c directory hierarchy in the user directory and compiles the **Actions.java** file there, thus making it part of the package. Note that the **package** statement is not recorded—you need to insert it manually.

Viewing the Java Methods

VuGen provides a navigator that lets you view all of the Java classes and methods in your application's packages.



To insert a class or method into your script, you select it and paste it into your script. For step-by-step instructions, see "Manually Inserting Java Methods" on page 472.

The lower part of the dialog box displays a description of the Java object, its prototype, return values and path. In the following example, the description indicates that the deserialize method is a public static method that receives two parameters—a string and an integer. It returns a java.lang.object and throws an exception.

```
public static synchronized java.lang.Object deserialize (java.lang.String, int) throws
Exception
```

The following table describes the icons that represent the various Java objects:

Icon	Item	Example
	Package	java.util
	Class	public class Hashtable extends java.util.Dictionary implements java.lang.Cloneable, java.io.Serializable
	Interface Class (gray icon)	public interface Enumeration
	Method	public synchronized java.util.Enumeration keys ()
	Static Method (yellow icon)	public static synchronized java.util.TimeZone getTimeZone
	Constructor Method	public void Hashtable ()

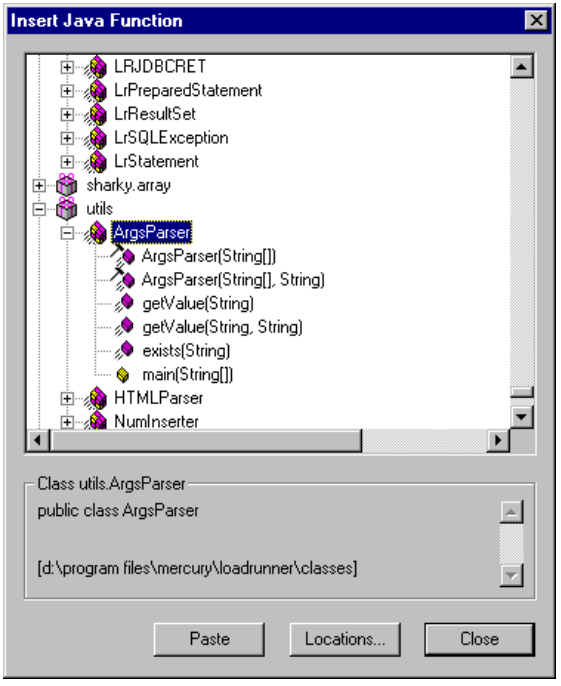
Manually Inserting Java Methods

You use the Java Function navigator to view and add Java functions to your script. The following section apply to EJB Testing and Java Record/Replay Vusers. You can customize the function generation settings by modifying the configuration file. For more information, see "Configuring Script Generation Settings" on page 474.

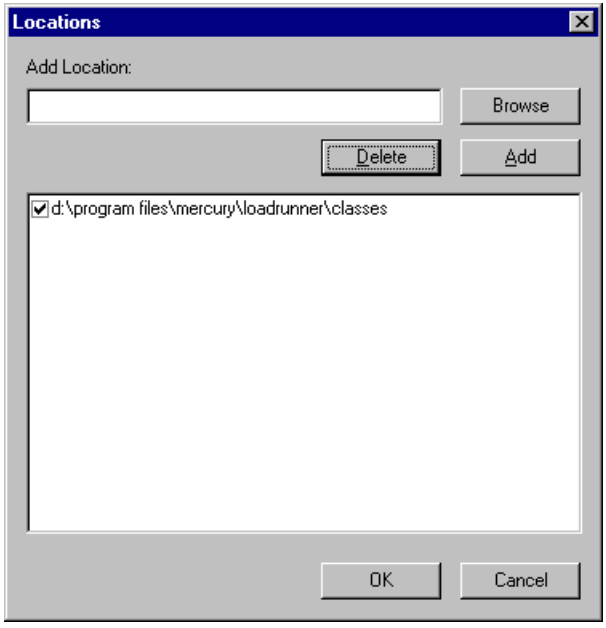
To insert Java functions:

1 Click within your script at the desired point of insertion. When you paste a function, VuGen places it at the location of the cursor.

2 Select **Insert** > **Insert Java Function**. The Insert Java Function dialog box opens.



3 Click **Locations**. The Locations dialog box opens. By default, VuGen lists the paths defined in the CLASSPATH environment variable.



4 Click **Browse** to add another path or archive to the list. To add a path, select **Browse** > **Folder**. To add an archive (**jar** or **zip**), select **Browse** > **File**. When you select a folder or a file, VuGen inserts it in the **Add Location** box.

5 Click **Add** to add the item to the list.

6 Repeat steps 4 and 5 for each path or archive you want to add.

7 Select or clear the check boxes to the left of each item in the list. If an item is checked, its members will be listed in the Java Class navigator.

8 Click **OK** to close the Locations dialog box and view the available packages.

9 Click the plus and minus signs to the left of each item in the navigator, to expand or collapse the trees.

10 Select an object and click **Paste**. VuGen places the object at the location of the cursor in the script. To paste all the methods of a class into your script, select the class and click **Paste**.

11 Repeat the previous step for all of the desired methods or classes.

12 Modify the parameters of the methods. If the script generation setting **DefaultValues** is set to **true**, you can use the default values inserted by VuGen. If **DefaultValues** is set to **false**, you must add parameters for all methods you insert into the script.

In addition, modify any return values. For example, if your script generated the following statement "(String)=LavaVersion.getVersionId();", replace (String) with a string type variable.

13 Add any necessary statements to your script such as imports or Vuser API Java functions (described in Chapter 38, "Programming Java Scripts").

14 Save the script and run it from VuGen.

Configuring Script Generation Settings

You can customize the way the navigator adds methods to your script in the following areas:

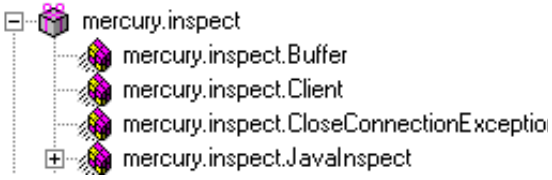
- Class Name Path
- Automatic Transactions
- Default Parameter Values
- Class Pasting

To view the configuration setting, open the **jquery.ini** file in VuGen's dat directory.

```
[Display]
FullClassName=False

[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

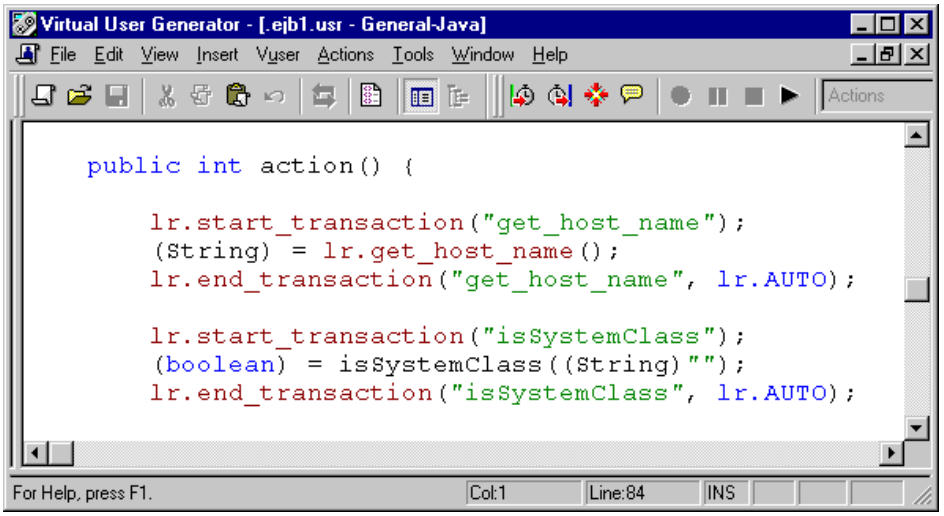
Class Name Path

The **FullClassName** option displays the complete package and class name in the Java Function navigator. This option does not affect the way the functions are added into the script—it only affects the way the classes are displayed in the navigator. By default, this option is set to false. If your packages have many classes and you are unable to view the package and class names at the same time, you should enable this option.

FullClassName enabled	FullClassName disabled
	

Automatic Transactions

The **AutoTransaction** setting creates a Vuser transaction for all methods. When you enable this option, VuGen automatically encloses all Java methods with **lr.start_transaction** and **lr.end_transaction** functions. This allows you to individually track the performance of each method. This option is disabled by default.



Default Parameter Values

The **DefaultValues** setting includes default values for all methods you paste into your script. This option is enabled by default and inserts a null for all objects. If you disable this option, you must manually insert parameter values for all functions in the script. The following table illustrates the DefaultValues flag enabled and disabled.

DefaultValues enabled	DefaultValues disabled
lr.message((String)"");	lr.message((String));
lr.think_time((int)0);	lr.think_time((int));
lr.enable_redirection((boolean)false);	lr.enable_redirection((boolean));
lr.save_data((byte[])null, (String)"");	lr.save_data((byte[]), (String));

Class Pasting

The **CleanClassPaste** setting pastes a class so that it will compile cleanly: with an instance returning from the constructor, with default values as parameters, and without a need for import statements. Using this option, you will most likely be able to run your script without any further modifications. If you disable this option (default), you may need to manually define parameters and include import statements. Note that this setting is only effective when you paste an entire class into your script—not when you paste a single method.

The following segment shows the toString method pasted into the script with the CleanClassPaste option enabled.

```
_class.toString();
    // Returns: java.lang.String
```

The same method with the CleanClassPaste option disabled is pasted as follows:

```
(String) = toString();
```

The next segment shows the **NumInserter** Constructor method pasted into the script with the CleanClassPaste option enabled.

```
utils.NumInserter _numinserter = new utils.NumInserter
            ((java.lang.String)"", (java.lang.String)"", (java.lang.String)""…);
// Returns: void
```

The same method with the CleanClassPaste option disabled is pasted as:

```
new utils.NumInserter((String)"", (String)"", (String)"",...);
```

Java Custom Filters

When testing your Java application, your goal is to determine how the server reacts to client requests. When load testing, you want to see how the server responds to a load of many users. With VuGen's Java Vuser, you create a script that emulates a client communicating with your server.

VuGen provides filter files that define hooking properties for commonly used methods. There are filter definitions for RMI, CORBA, JMS, and JACADA protocols. You can also define custom filters as described below.

When you record a method, the methods which are called from the recorded method either directly or indirectly, will not be recorded.

In order to record a method, VuGen must recognize the object upon which the method is invoked, along with the method's arguments. VuGen recognizes an object if it is returned by another recorded method provided that:

- the construction method of that object is hooked
- it is a primitive or a built-in object
- It supports a serializable interface.

You can create a custom filter to exclude unwanted methods. When recording a Java application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters for RMI, CORBA, JMS, and JACADA protocols were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your JAVA application's calls or exclude unnecessary calls. Custom JAVA protocols, proprietary enhancements and extensions to the default protocols, and data abstraction all require a custom filter definition.

Guidelines for Setting Filters

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, VuGen allows you to record with a stack trace that logs all of the methods that were called by your application. In order to record with stack trace set the log level to **Detailed**. For more information, see "Defining an Effective Filter" on page 481.

Once you determine the required methods and classes, you include them by updating the **user.hooks** file. When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Note: If you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this, is that if you rerecord a script after modifying the filters, it will overwrite all manual changes.

Determining which Elements to Include or Exclude

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- **Top Down Approach.** An approach in which you include the relevant package and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined layer which implements all client-server activity without involving any GUI elements.
- **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT packages and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- If you identify a non-client/server call in your script, exclude its method in the filter.
- During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen serializes it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by deserializing it.
- VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the **lr.deserialize()** method in your script. For more information see "Using the Serialization Mechanism" on page 492.
- Exclude all activity which involves GUI elements.

➤ Add classes for utilities that may be required for the script to be compiled.

Defining an Effective Filter

When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

To define an effective filter:

1. Create a new filter based on one of the built-in filters by modifying the **user.hooks** file which is located in the product's **classes** directory.
2. Open the Recording Options (Ctrl+F7) and select the **Log Options** node. Select the Log Level to **Detailed**.
3. Record your application. Click **Start Record** (Ctrl + R) to begin and **Stop** (Ctrl + F5) to end.
4. View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain and correlate, you should customize the script's filter.
5. Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) or by viewing a Stack Trace of the script.
6. Set the filter to include the relevant methods. For more information, see "Determining which Elements to Include or Exclude" on page 480.
7. Record the application again. You should always rerecord the application after modifying the filter.
8. Repeat steps 4 through 7 until you get a simple script which can be easily maintained and correlated.
9. Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information about the built-in correlation mechanism, see Chapter 27, "Java - Correlating."

Note: Do not modify any of the other .hooks file as it might damage the VuGen recorder.

Adding custom hooks to the default recorder is a complicated task and should be considered thoroughly as it has both functional and performance consequences.

Incorrect hooking definitions can lead to incorrect scripts, slow recording, and application freeze-up.

Hooks Files Structure

The following section describes the structure of a typical hooks file:

```
[Hook-Name]
class      = MyPackage.MyClass
method     = MyMethod
signature  = ()V
ignore_cl =
ignore_mtd =
ignore_tree =
cb_class   = mercury.ProtocolSupport
cb_mtd =
general_cb = true
deep_mode = soft | hard
make_methods_public = true | false
lock = true | false
```

The hook files are structured as .ini files where each section represents a hook definition. Regular expressions are supported in some of the entries. Any entry that uses regular expression must start with a '!'.

Hook-Name

Specifies the name of this section in the hooks file. Hook-Name must be unique across all hooks files. A good practice is to give the fully qualified class name and method. For example:

```
[javax.jms.Queue.getQueueName]
```

Class

A fully qualified class name. Regular expression can be used to include several classes from the same package, a whole package, several packages, or any class that matches a name. For example:

```
Class = !javax\.jms\.*
```

Method

The simple name of the method to include. Regular expressions can be used to include more than one method from the class. For example:

```
Method = getQueueName
```

Signature

The standard Java internal type signature of the method. To determine the signature of a method, run the command `javap -s class-name` where `class name` is the fully qualified name of the class. Regular expressions can be used to include several methods with the same name, but with different arguments. For example:

```
Signature = !.*
```

ignore_cl

A specific class to ignore from the classes that match this hook. This can be a list of comma separated class names. Each item in the list can contain a regular expression. If an item in the list contains a regular expression, prepend a '!' to the class name. For example:

```
Ignore_cl = !com.hp.jms.Queue,!com\.hp\..*
```

ignore_mtd

A specific method to ignore. When the loaded class method matches this hook definition, this method will not be hooked. The method name must be the simple method name followed by the signature (as explained above). To ignore multiple methods, list them in a comma separated list. To use a regular expression, prepend a '!' to the method name. For example:

```
Ignore_cl = open, close
```

ignore_tree

A specific tree to ignore. When the name of the class matches the ignore tree expression, any class that inherits from it will not be hooked, if it matches this hooks definition. To ignore multiple trees, list them in a comma separated list. To use a regular expression, prepend a '!' to the class name. This option is relevant only for hooks that are defined as deep.

cb_class

The callback class that gets the call from the hooked method. It should always be set to **mercury.ProtocolSupport**.

cb_mtd

A method in the callback class that gets the call from the hooked method. If omitted, it uses the default, **general_rec_func**. For cases where you just need to lock the subtree of calls, use **general_func** instead.

general_cb

The general callback method. This value should always be set to **true**.

Deep_mode

Deep mode refers to classes and interfaces that inherit or implement the class or interface that the hook is listed for. The inherited classes will be hooked according to the type of hook: **Hard**, **Soft**, or **Off**.

- **Hard.** Hooks the current class and any class that inherits from it. If regular expressions exist, they are matched against every class that inherits from the class in the hook definition. Interface inheritance is treated the same as class inheritance.
- **Soft.** Hooks the current class and any class that inherits from it, only if the methods are overridden in the inheriting class. If the hook lists an interface, then if a class implements this interface those methods will be hooked. If they exist in classes that directly inherit from that class they will also be hooked. However, if the hook lists an interface and a class implements a second interface that inherits from this interface, the class will not be hooked.

Note: Regular expressions are not inherited but converted to actual methods.

➤ **Off.** Only the class listed in the hook definition and the direct inheriting class will be hooked. If the hook lists an interface, only classes that directly implement it will be hooked.

make_methods_public:

Any method that matches the hook definition will be converted to public. This is useful for custom hooks or for locking a sub tree of calls from a non-public method.

Note that this applies only during record. During replay, the method will use the original access flags. In the case of non-public methods, it will throw java.lang.VerifyError.

Lock

When set to **true**, it locks the sub tree and prevents the calling of any method originating from the original method.

When set to **false**, it will unlock the sub tree, record any method originating from the current method (if it is hooked), and invoke the callback.

27

Java - Correlating

VuGen's correlation allows you to link Java Vuser functions by using the results of one statement as input to another.

This chapter includes:

About Correlating Java Scripts

Vuser scripts containing Java code often contain dynamic data. When you record a Java Vuser script, the dynamic data is recorded into scripts, but cannot be re-used during replay. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. In many cases, correlation will solve the problem by enabling you to use the results of one statement as input to another.

VuGen's Java recorder attempts to automatically correlate statements in the generated script. It only performs correlation on Java objects. When it encounters a Java primitive (byte, character, boolean, integer, float, double, short, and long) during recording, the argument values appear in the script without association to variables. VuGen automatically correlates all objects, arrays of objects, and arrays of primitives. Note that Java arrays and strings are also considered objects.

VuGen employs several levels of correlation: Standard, Enhanced, Strings. You enable or disable correlation from the Recording options. An additional method of Serialization can be used to handle scripts where none of the former methods can be applied. For more information, see "Using the Serialization Mechanism" on page 492.

Standard Correlation

Standard correlation refers to the automatic correlation performed during recording for simple objects, excluding object arrays, vectors, and container constructs.

When the recorded application invokes a method that returns an object, VuGen's correlation mechanism records these objects. When you run the script, VuGen compares the generated objects to the recorded objects. If the objects match, the same object is used. The following example shows two CORBA objects my_bank and my_account. The first object, my_bank, is invoked; the second object, my_account, is correlated and passed as a parameter in final line of the segment:

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {

      Bank my_bank = bankHelper.bind("bank", "shunra");
      Account my_account = accountHelper.bind("account","shunra");

      my_bank.remove_account(my_account);
     }
:
}
```

Advanced Correlation

Advanced or **deep** correlation refers to the automatic correlation performed during recording for complex objects, such as object arrays and CORBA container constructs.

The deep correlation mechanism handles CORBA constructs (structures, unions, sequences, arrays, holders, 'any's) as containers. This allows it to reference inner members of containers, additional objects, or different containers. Whenever an object is invoked or passed as a parameter, it is also compared against the inner members of the containers.

In the following example, VuGen performs deep correlation by referencing an element of an array. The remove_account object receives an account object as a parameter. During recording, the correlation mechanism searches the returned array my_accounts and determines that its sixth element should be passed as a parameter.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_banks[] = bankHelper.bind("banks", "shunra");
    my_accounts[] = accountHelper.bind("accounts","shunra");

    my_banks[2].remove_account(my_accounts[6]);
  }
:
}
```

The following segment further illustrates enhanced correlation. The script invokes the send_letter object that received an address type argument. The correlation mechanism retrieves the inner member, address, in the sixth element of the my_accounts array.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_banks = bankHelper.bind("bank", "shunra");
    my_accounts = accountHelper.bind("account", "shunra");

    my_banks[2].send_letter(my_accounts[6].address);
  }
:
}
```

String Correlation

String correlation refers to the representation of a recorded value as an actual string or a variable. When you disable string correlation (the default setting), the actual recorded value of the string is indicated explicitly within the script. When you enable string correlation, it creates a variable for each string, allowing you to use it at a later point in the script.

In the following segment, string correlation is enabled—you store the value returned from the get_id method in a string type variable for use later on in the script.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_bank = bankHelper.bind("bank", "shunra");
    my_account1 = accountHelper.bind("account1", "shunra");
    my_account2 = accountHelper.bind("account2", "shunra");

    string = my_account1.get_id();
    string2 = my_account2.get_id();
    my_bank.transfer_money(string, string2);
  }
:
}
```

You set the correlation method from the **Correlation** tab in the recording options.

- **Correlate Strings.** Correlate strings in script during recording. If you disable this option, the actual recorded values are included in the script between quotation marks. If this option is disabled, all other correlation options are ignored (disabled by default).
- **Correlate String Arrays.** Correlate strings within string arrays during recording. If you disable this option, strings within arrays are not correlated and the actual values are placed in the script (enabled by default).

- **Advanced Correlation.** Enables correlation on complex objects such as arrays and CORBA container constructs and arrays. This type of correlation is also known as deep correlation (enabled by default).
- **Correlation Level.** Determines the level of deep correlation—how many inner containers to search.
- **Correlate Collection Type.** Correlate objects contained in a Collection class for JDK 1.2 or higher (disabled by default).

Using the Serialization Mechanism

In RMI, and some cases of CORBA, the client AUT creates a new instance of a Java object using the java.io.serializable interface. It passes this instance as a parameter for a server invocation. In the following segment, the instance p is created and passed as a parameter.

```
// AUT code:
java.awt.Point p = new java.awt.Point(3,7);
map.set_point(p);
:
```

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user directory. Additional parameters are saved as new binary data files, numbered sequentially. VuGen generates the following code:

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
        map.set_point(p);
     }
:
}
```

The integer passed to **lr.deserialize** represents the number of binary data files in the Vuser directory.

To parameterize the recorded value, use the public setLocation method (for information, see the JDK function reference). The following example uses the setLocation method to set the value of the object, p.

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
        p.setLocation(2,9);
        map.set_point(p);
     }
:
:
}
```

In certain instances the public method of **setLocation** is not applicable. As an alternative, you can use the API of your class that incorporate get or set accessor methods. If you are working with AUT classes that do not have get/set methods or use private methods, or if you are unfamiliar with the classes' API, you can use VuGen's built-in serialization mechanism. This mechanism allows you to expand objects in their ASCII representation and manually parameterize the script. You enable this mechanism in the Recording Options dialog box (see Chapter 75, "Java Recording Options"for more information).

VuGen generates an **lr.deserialize** method that deserializes the data or displays complex data structures as serial strings. Once the structure is broken down to its components, it is easier to parameterize. The **lr.deserialize** method receives two arguments, a string and an integer. The string is the parameter's value that is to be substituted during replay. The integer is the index number of binary file to load.

If you choose not to expand objects in your script by clearing the Unfold Serialized Objects check box, you can control the serialization mechanism by passing arguments to the lr.deserialize method. The first argument is an integer indicating the number of binary files to load. The second integer is a boolean value:

true	Use VuGen's serialization mechanism.
false	Use the standard Java serialization mechanism.

The following segment shows a generated script in which the serialization mechanism was enabled.

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        _string = "java.awt.Point __CURRENT_OBJECT = {" +
                  "int x = "#5#" +
                  "int y = "#8#" +
        "}";
        java.awt.Point p = (java.awt.Point)lr.deserialize(_string,0);
        map.set_point(p);
     }
:
}
```

The string values are placed between delimiters.The default delimiter is "#". You can change the delimiter in the **Serialization** tab of the recording options. Delimiters are used to speed up the parsing of the string during replay.

When modifying the string, you must maintain the following rules:

- Order of lines may not be changed. The parser reads the values one-by-one—not the member names.
- Only values between two delimiters may be modified.
- Object references may not be modified. Object references are indicated only to maintain internal consistency.

- "_NULL_" can appear as a value, representing the Java null constant. You can replace it with string type values only.
- Objects may be deserialized anywhere in the script. For example, you can deserialize all objects in the **init** method and use the values in the **action** method.
- Maintain internal consistency for the objects. For example, if a member of a vector is **element count** and you add an element, you must modify the element count.

In the following segment, a vector contains two elements:

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
         _string = "java.util.Vector CURRENTOBJECT = {" +
                   "int capacityIncrement = "#0#" +
                   "int elementCount = #2#" +
                   "java/lang/Object elementData[] = {" +
                         "elementData[0] = #First Element#" +
                         "elementData[1] = #Second Element#" +
                         "elementData[2] = _NULL_" +
                          ....
                         "elementData[9] = _NULL_" +

                   "}" +
         "}";
         _vector = (java.util.Vector)lr.deserialize(_string,0);
         map.set_vector(_vector);
     }
:
}
```

In the following example, one of the vector's elements was changed—a "_NULL_" value was changed to "Third element". In coordination with the addition of the new element, the "elementCount" member was modified to "3".

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        _string = "java.util.Vector CURRENTOBJECT = {" +
                  "int capacityIncrement = "#0#" +
                  "int elementCount = #3#" +
                  "java/lang/Object elementData[] = {" +
                        "elementData[0] = #First Element#" +
                        "elementData[1] = #Second Element#" +
                        "elementData[2] = #Third Element#" +
                          ....
                        "elementData[9] = _NULL_" +
                  "}" +
        "}";
        _vector = (java.util.Vector)lr.deserialize(_string,0);
        map.set_vector(_vector);
     }
:
}
```

Due to the complexity of the serialization mechanism, which opens up the objects to ASCII representation, opening large objects while recording may increase the time required for script generation. To decrease this time, you can specify flags which will improve the performance of the serialization mechanism.

When adding **lr.deserialize** to your script, we recommend that you add it to the **init** method—not the **action** method. This will improve performance since VuGen will only deserialize the strings once. If it appears in the **action** method, VuGen will deserialize strings for every iteration.

The following list shows the available options which you set in **Serialization** tab of the recording options:

- Serialization Delimiter
- Unfold Serialized Objects
- Unfold Arrays
- Limit Array Entries
- Ignore Serialized Objects

See Chapter 75, "Java Recording Options"for complete information on the recording options.

28

Enterprise Java Beans (EJB) Protocol

VuGen helps you create a script for testing Enterprise Java Beans (EJB) objects on your application server.

This chapter includes:

About EJB Testing

VuGen provides several tools for developing a script that tests Java applications. For generating a Vuser script through recording, use the Java Record and Replay Vuser. For creating a script through programming, use the Custom Java Vuser type.

EJB Testing Vusers differ from the standard Java Vusers in that VuGen automatically creates a script to test or tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. VuGen's EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to test or tune, and VuGen generates a script that emulates each of the EJB's methods.

It creates transactions for each method so that you can measure its performance and locate problems. In addition, each method is wrapped in a **try and catch** block for exception handling.

Note that in order to create EJB testing scripts, the EJB Detector must be installed and active on the application server host. The Detector is described in the following sections.

VuGen also has a built-in utility for inserting methods into your script. Using this utility, you display all of the available packages, select the desired methods, and insert them into your script. For more information, see "Running EJB Vuser Scripts" on page 515.

Working with the EJB Detector

The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

Installing the EJB Detector

The EJB Detector can be installed and invoked on the application server's machine or alternatively, on the client machine. To run the EJB Detector on the client machine you must have a mounted drive to the application server machine.

To install the EJB detector agent:

1. Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
2. Unzip the **<LR_root>\ejbcomponent\ejbdetector.jar** file into the EJB Detector directory.

Running the EJB Detector

The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

The EJB Detector can run from the command-line, or from a batch file.

To run the EJB Detector from the command line:

1. Before running the EJB Detector from the command line, add the DETECTOR_HOME\classes and the DETECTOR_HOME\classes\xerces.jar to the CLASSPATH environment variable.
2. If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the CLASSPATH:

   For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar

   For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar
3. If your EJBs use additional classes directory or .jar files, add them to the CLASSPATH.

4 To run the EJB Detector from the command-line, use the following string:

```
java EJBDetector [search root dir] [listen port]
```

search root dir	One or more directories or files in which to search for EJBs (separated by semicolons). Follow these guidelines: **BEA WebLogic Servers 4.x and 5.x.** Specify the application server root directory. **BEA WebLogic Servers 6.x.** Specify full path of the domain folder. **WebSphere Servers 3.x.** Specify the full path of the deployed EJBs folder. **WebSphere Servers 4.0.** Specify the application server root directory. **Oracle OC4J.** Specify the application server root directory. **Sun J2EE Server.** Specify the full path to the deployable **.ear** file or directory containing a number of **.ear** files. If unspecified, the classpath will be searched.
listen port	The listening port of the EJB Detector.The default port is 2001. If you change this port number, you must also specify it in the **Host name** box of the **Generate EJB Test** dialog box. For example, if your host is metal, if you are using the default port, you can specify **metal**. If you are using a different port, for example, port 2002, enter **metal:2002**.

To run the EJB Detector from a batch file:

You can launch the EJB detector using a batch file, **EJB_Detector.cmd**. This file resides in the root directory of the EJB Detector installation, after you unzip **ejbdetector.jar**.

1 Open env.cmd in the EJB Detector root directory, and modify the following variables according to your environment:

JAVA_HOME	the root directory of JDK installation
DETECTOR_INS_DIR	the root directory of the Detector installation

APP_SERVER_DRIVE	the drive hosting the application server installation
APP_SERVER_ROOT	Follow these guidelines: **BEA WebLogic Servers 4.x and 5.x.** Specify the application server root directory. **BEA WebLogic Servers 6.x.** Specify full path of the domain folder. **WebSphere Servers 3.x.** Specify the full path of the deployed EJBs folder. **WebSphere Servers 4.0.** Specify the application server root directory. **Oracle OC4J.** Specify the application server root directory. **Sun J2EE Server.** Specify the full path to the deployable **.ear** file or directory containing a number of **.ear** files.
EJB_DIR_LIST (optional)	list of directories/files, separated by ';' and containing deployable .ear/.jar files, and any additional classes directory or .jar files or used by your EJBs under test.

2 Save **env.cmd**.

3 If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested, as well as the following vendor EJB classes, to the CLASSPATH in the env file:

- For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar
- For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar

4 Run the EJB_Detector.cmd or EJB_Detector.sh (Unix platforms) batch file to collect information about the deployable applications containing EJBs, for example:

```
C:\>EJB_Detector [listen_port]
```

where listen_port is an optional argument specifying a port number on which the EJB Detector will listen for incoming requests (default is 2001).

EJB Detector Output and Log Files

You can examine the output of the EJB Detector to see if it has detected all the active EJBs. The output log shows the paths being checked for EJBs. At the end of the scan, it displays a list of the EJBs that were found, their names and locations. For example:

```
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_beanManaged.jar…
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_statefulSession.jar…
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_statelessSession.jar…
------------------------ Found 3 EJBs --------------------------
** PATH: f:/weblogic/myserver/ejb_basic_beanManaged.jar
- BEAN: examples.ejb.basic.beanManaged.AccountBean
** PATH: f:/weblogic/myserver/ejb_basic_statefulSession.jar
- BEAN: examples.ejb.basic.statefulSession.TraderBean
** PATH: f:/weblogic/myserver/ejb_basic_statelessSession.jar
- BEAN: examples.ejb.basic.statelessSession.TraderBean
```

If no EJBs were detected (that is, "Found 0 EJBs"), check that the EJB jar files are listed in the "Checking EJB Entry:…" lines. If they are not listed, check that the **search root dir** path is correct. If they are being inspected but still no EJBs are detected, check that these EJB jar files are deployable (can be successfully deployed into an application server). A deployable jar file should contain the Home Interface, Remote Interface, Bean implementation, the Deployment Descriptor files (xml files, or .ser files), and additional vendor-specific files.

If you still encounter problems, set the debug properties in the **detector.properties** file, located in the DETECTOR_HOME\classes directory, to retrieve additional debug information.

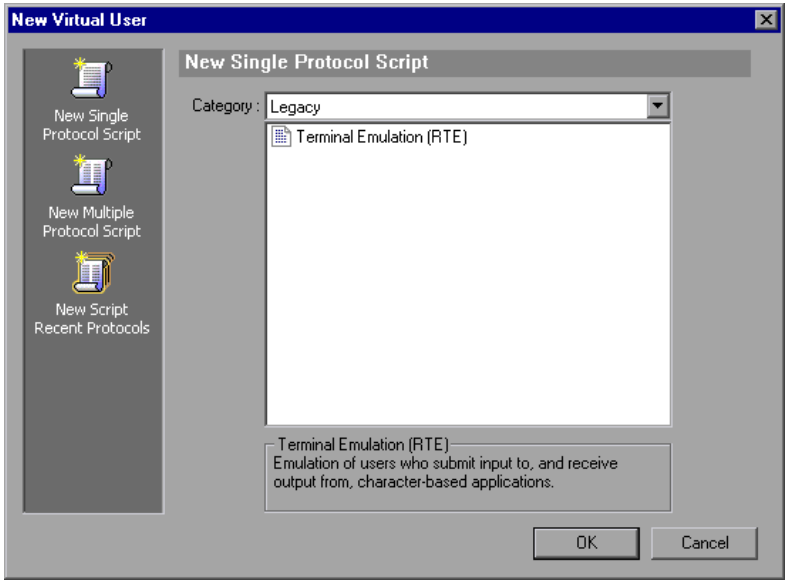
After the EJBs are detected, the HTTP Server is initialized and waits for requests from the VuGen EJB-Testing Vuser. If there are problems in this communication process, enable the property webserver.enableLog in the **webserver.properties** file located in the DETECTOR_HOME\classes directory.

This enables printouts of additional debug information, and other potentially important error messages in the **webserver.log** file.

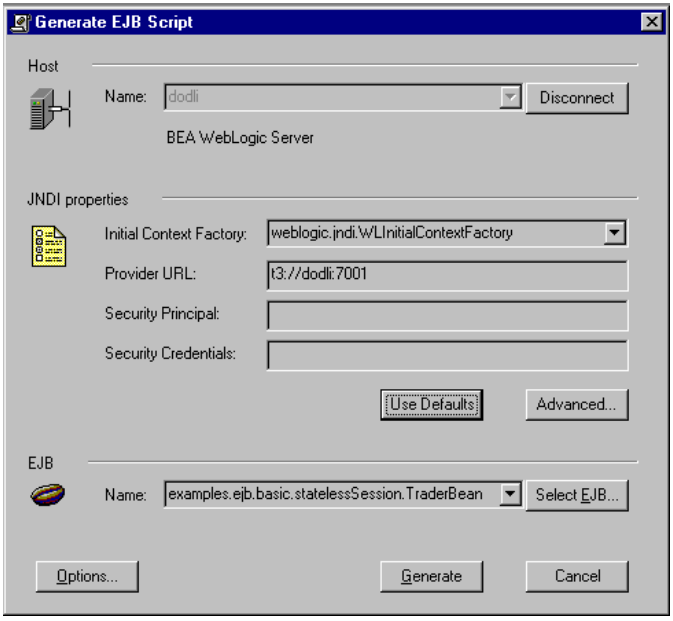
Creating an EJB Testing Vuser

To create an EJB Vuser script:

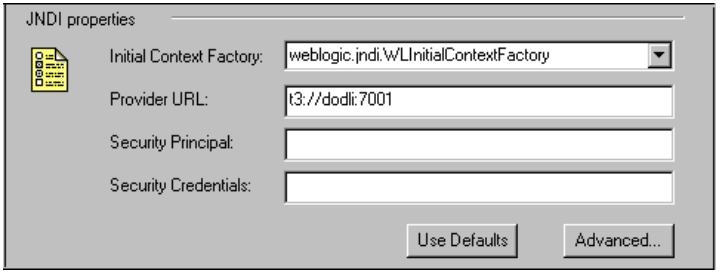
1 Select **File** > **New** or click the **New** button. The New Virtual User dialog box opens.



2 Select **EJB Testing** from the **Enterprise Java Beans** category and click **OK**. VuGen opens a blank Java Vuser script and opens the Generate EJB Script dialog box.



3 Specify a machine on which VuGen's EJB Detector is installed. Note that the Detector must be running in order to connect. Click **Connect**. The **JNDI properties** section is enabled.



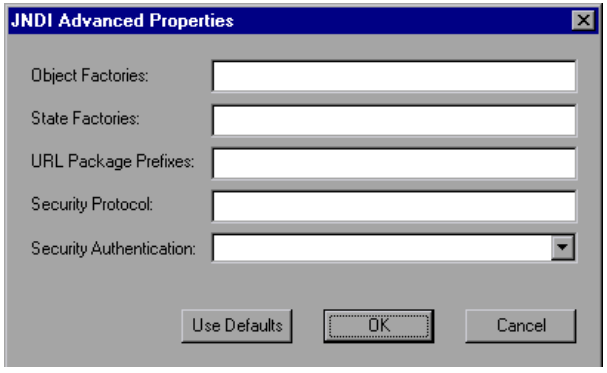
4 The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory and** the **Provider URL**.

If your application server requires authentication, enter the user name in the **Security Principal** box and a password in the **Security Credentials** box.

Here are the default values of the two JNDI mandatory properties:

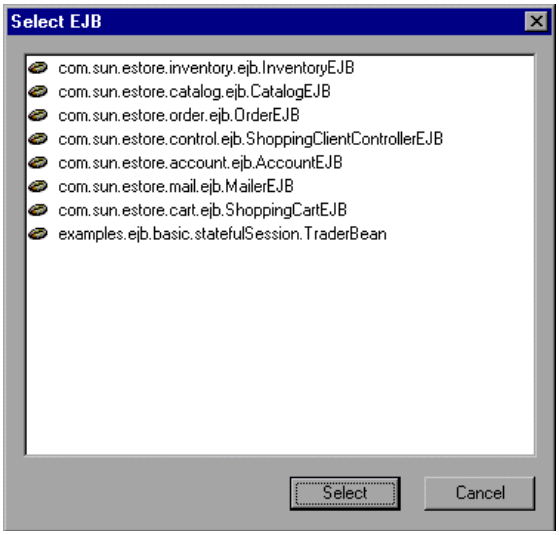
Type	Initial Context Factory	Provider URL
WebLogic	weblogic.jndi.WLInitialContextFactory	t3://<appserver_host>:7001
WebSphere 3.x	com.ibm.ejs.ns.jndi.CNInitialContextFactory	iiop://<appserver_host>:900
WebSphere 4.x	com.ibm.websphere.naming.WsnInitial ContextFactory	iiop://<appserver_host>:900
Sun J2EE	com.sun.enterprise.naming. SerialInitContextFactory	N/A
Oracle	com.evermind.server. AppplicationClientInitialContextFactory	ormi://<appserver_host>/ <application_name> (the app. name of the EJB in <oc4j>/config/server.xml)

5 To set advanced properties for the JNDI, click **Advanced** to open the JNDI Advanced Properties dialog box.



Specify the desired properties: **Object Factory**, **State Factory**, **URL Package Prefixes**, **Security Protocol**, and **Security Authentication**. Click **OK**.

6 In the **EJB** section of the dialog box, click **Select** to select the EJB for which you want to create a test. A dialog box opens with a list of all the EJBs currently available to you from the application server.



7 Highlight the EJB you want to test and click **Select**.

8 In the Generate EJB Script dialog box, click **Generate**. VuGen creates a script with Java Vuser functions. The script contains code that connects to the application server and executes the EJB's methods.

9 Save the script.

Note that you cannot generate test code for an additional EJB, within an existing script. To create a test for another EJB, open a new script and repeat steps 2-9.

Understanding EJB Vuser Scripts

VuGen generates a script that tests your EJB, based on the JNDI (Java Naming and Directory Interface) properties you specified when creating the Vuser script. JNDI is Sun's programming interface used for connecting Java programs to naming and directory services such as DNS and LDAP.

Each EJB Vuser script contains three primary parts:

- Locating the EJB Home Using JNDI
- Creating an Instance
- Invoking the EJB Methods

Locating the EJB Home Using JNDI

The first section of the script contains the code that retrieves the JNDI properties. Using the specified context factory and provider URL, it connects to the application server, looks up the specified EJB and locates the EJB Home.

In the following example, the JNDI Context Factory is weblogic.jndi.WLInitialContextFactory, the URL of the provider is t3://dod:7001 and the JNDI name of the selected EJB is carmel.CarmelHome.

```
public class Actions
{

    public int init() {
        CarmelHome _carmelhome = null;
        try {
            // get the JNDI Initial Context
            java.util.Properties p = new java.util.Properties();
            p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
            p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001");
            javax.naming.InitialContext _context = new javax.naming.InitialContext(p);

            // lookup Home Interface in the JNDI context and narrow it
            Object homeobj = _context.lookup("carmel.CarmelHome");
            _carmelhome =
(CarmelHome)javax.rmi.PortableRemoteObject.narrow(homeobj, CarmelHome.class);

        } catch (javax.naming.NamingException e) {
            e.printStackTrace();
        }
```

Note: If the script is generated with an EJB Detector running on the client rather than an application server, you must manually modify the URL of the provider. For example, in the following line, the provider specifies dod as the EJB detector host name:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001")
```

Replace the recorded host name with the application server name, for example:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://bealogic:7001")
```

You can specify the provider URL before recording, so you don't have to modify it manually, in the **JDNI Properties** section of the Generate EJB Script dialog.

Creating an Instance

Before executing the EJB methods, the script creates a Bean instance for the EJB. The creation of the instance is marked as a transaction to allow it to be analyzed after the script is executed. In addition, the process of creating an instance is wrapped in a **try and catch** block providing exception handling.

For Session Beans. Use the EJB home 'create' method to get a new EJB instance.

In the following example, the script creates an instance for the Carmel EJB.

```
// create Bean instance
Carmel _carmel = null;
try {
    lr.start_transaction("create");
    _carmel = _carmelhome.create();
    lr.end_transaction("create", lr.AUTO);
} catch (Throwable t) {
    lr.end_transaction("create", lr.FAIL);
    t.printStackTrace();
}
```

For Entity Beans - use the findByPrimaryKey method to locate the EJB instance in an existing database, and if not found, then use the create method, to create it there.

In the following example, the script attempts to locate an instance for the account EJB, and if it fails then creates it.

```
// find Bean instance
try {
    com.ibm.ejs.doc.account.AccountKey _accountkey = new
com.ibm.ejs.doc.account.AccountKey();
    _accountkey.accountId = (long)0;

    lr.start_transaction("findByPrimaryKey");
    _account = _accounthome.findByPrimaryKey(_accountkey);
    lr.end_transaction("findByPrimaryKey", lr.AUTO);
} catch (Throwable thr) {

    lr.end_transaction("findByPrimaryKey", lr.FAIL);
    lr.message("Couldn't locate the EJB object using a primary key. Attempting to
manually create the object... ["+thr+"]");

    // create Bean instance
    try {
        lr.start_transaction("create");
        _account = _accounthome.create((com.ibm.ejs.doc.account.AccountKey)null);
        lr.end_transaction("create", lr.AUTO);
    } catch (Throwable t) {
        lr.end_transaction("create", lr.FAIL);
        t.printStackTrace();
    }

}
```

You can use other find methods, supplied by your Entity Bean, to locate the EJB instance. For example:

```
// get an enumeration list of all Email EJB instances that represents
// the name 'John' in the database.
Enumeration enum = home.findByName("John");
    while (enum.hasMoreElements()) {
        Email addr = (Email)enum.nextElement();
        ...
}
```

Invoking the EJB Methods

The final part of the script contains the actual methods of the EJB. Each method is marked as a transaction to allow it to be analyzed after running the script. In addition, each method is wrapped in a try and catch block providing exception handling. When there is an exception, the transaction is marked as failed, and the script continues with the next method. VuGen creates a separate block for each of the EJB methods.

```
// ------- Methods ------------

        int _int1 = 0;
        try {
            lr.start_transaction("buyTomatoes");
            _int1 = _carmel.buyTomatoes((int)0);
            //lr.value_check(_int1, 0, lr.EQUALS);
            lr.end_transaction("buyTomatoes", lr.AUTO);
        } catch (Throwable t) {
            lr.end_transaction("buyTomatoes", lr.FAIL);
            t.printStackTrace();
        }
```

VuGen inserts default values for the methods, for example, 0 for an integer, empty strings ("") for Strings, and NULL for complex Java objects. If necessary, modify the default values within the generated script.

```
            _int1 = _carmel.buyTomatoes((int)0);
```

The following example shows how to change the default value of a non-primitive type using parameterization:

```
Detail details = new Details(<city>,<street>,<zip>,<phone>);
JobProfile job = new JobProfile(<department>,<position>,<job_type>);
Employee employee=new Employee(<first>,<last>, details, job, <salary>);
_int1 = _empbook.addEmployee((Employee)employee);
```

For methods that return a primitive, non-complex value or string, VuGen inserts a commented method lr.value_check. This method allows you to specify an expected value for the EJB method. To use this verification method, remove the comment marks (//) and specify the expected value. For example, the carmel.buyTomatoes method returns an integer.

```
_int1 = _carmel.buyTomatoes((int)0);
//lr.value_check(_int1, 0, lr.EQUALS);
```

If you expect the method to return a value of 500, modify the code as follows:

```
_int1 = _carmel.buyTomatoes((int)0);
lr.value_check(_int1, 500, lr.EQUALS);
```

If you want to check if the method does not return a certain value, modify the code as follows:

```
_int1 = _carmel.buyTomatoes((int)0);
lr.value_check(_int1, 10, lr.NOT_EQUALS);
```

If the expected value is not detected, an exception occurs and the information is logged in the output window.

```
System.err: java.lang.Exception: lr.value_check failed.[Expected:500 Actual:5000]
```

EJB Vuser scripts support all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes "//".

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. (see Run-Time settings) This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

Running EJB Vuser Scripts

After you generate a script for your EJB testing, and make the necessary modifications, you are ready to run your script. The EJB script allows you to perform two types of testing: functional and load. The functional testing verifies that the EJB, functions properly within your environment. The load testing allows you to evaluate the performance of the EJB when executing many users at one time.

To run a functional test:

1. Modify the default values that were automatically generated.
2. Insert value checks using the **lr.value_check** method. These methods were generated as comments in the script (see "Invoking the EJB Methods" on page 513).
3. Insert additional methods, and modify their default values. For more information, see the section on inserting Java functions in Chapter 25, "Java Protocols - Recording."
4. Set the general run-time settings for the script. For more information, see Chapter 79, "Configuring Run-Time Settings."
5. Set the Java VM run-time settings: Specify all additional classpaths and additional VM parameters. Make sure to include your application server EJB classes. The actual classes of the EJB under test are saved in the Vuser directory and retrieved automatically during replay. For information about specifying additional classpaths and setting the Java VM run-time settings, see Chapter 81, "Java and EJB Run-Time Settings".
6. For **Websphere 3.x** users:

   Using the IBM JDK 1.2 or higher:

   - Add the <WS>\lib\ujc.jar to the classpath.

   Using the Sun JDK 1.2.x:

   - Remove the file <JDK>\jre\lib\ext\iiimp.jar
   - Copy the following files from the <WS>\jdk\jre\lib\ext folder to the <JDK>\jre\lib\ext directory: iioprt.jar, rmiorb.jar.
   - Copy the 'ujc.jar' from the <WS>\lib folder, to <JDK>\jre\lib\ext.

➤ Copy the file <WS>\jdk\jre\bin\ioser12.dll to the <JDK>\jre\bin folder.

where <*WS*> is the home folder of the WebSphere installation and <*JDK*> is the home folder of the JDK installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

7 For **WebSphere 4.0** users:

Make sure that you have valid Java environment on your machine of IBM JDK1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<WS>/lib/webshpere.jar;
<WS>/lib/j2ee.jar;
```

Where <WS> is the home directory of the WebSphere installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

Note: If your application server is installed on a UNIX machine or if you are using WebSphere 3.0.x, you must install IBM JDK 1.2.x on the client machine to obtain the required files.

8 For **Oracle OC4J** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher (JDK1.3 preferable). Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<OC4J>/orion.jar;<OC4J>/ejb.jar;<OC4J>/jndi.jar; ;<OC4J>/xalan.jar;
<OC4J>/crimson.jar
```

where <OC4J> is the home folder of the application server installation.

9 For **Sun J2EE** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<J2EE>/j2ee.jar;<AppClientJar>
```

where <J2EE> is the home folder of the application server installation and <AppClientJar> is the full path to the application client jar file created automatically by the sdk tools during the deployment process.

10 For **WebLogic 4.x - 5.x** Users:

Make sure that you have valid Java environment on your machine (path & classpath). Open the Run-Time Settings dialog box and select the Java VM node. Add the following two entries to the Additional Classpath section:

```
<WL>/classes;<WL>/lib/weblogicaux.jar
```

where <*WL*> is the home folder of the WebLogic installation.

11 For **WebLogic 6.x** and **7.0** users:

Make sure that you have valid Java environment on your machine (path & classpath). WebLogic 6.1 requires JDK 1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entry to the Additional Classpath section:

```
<WL>/lib/weblogic.jar;    // Weblogic 6.x
<WL>/server/lib/weblogic.jar    // Weblogic 7.x
```

where <*WL*> is the home folder of the WebLogic installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

12 Run the script. Click the **Run** button or select **Vuser** > **Run**. View the Execution Log node to view any run-time errors. The execution log is stored in the **mdrv.log** file in the script's folder. A Java compiler (Sun's javac), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

29

Citrix Protocol

VuGen allows you to record the actions of a Citrix client communicating with its server using the Citrix ICA protocol. The resulting script is called a Citrix Vuser script.

The optional **Citrix Agent** helps you create an intuitive script that provides built-in synchronization. For more information, see Chapter 30, "Citrix - Agent for Citrix Presentation Server." See "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 538 for valuable tips on creating scripts.

This chapter includes:

About Creating Citrix Vuser Scripts

Citrix Vuser scripts emulate the Citrix ICA protocol communication between a Citrix client and server. VuGen records all activity during the communication and creates a Vuser script.

When you perform actions on the remote server, VuGen generates functions that describe these actions. Each function begins with a **ctrx** prefix. These functions emulate the analog movements of the mouse and keyboard. In addition, the **ctrx** functions allow you to synchronize the replay of the actions, by waiting for specific windows to open.

VuGen also allows you to record a Citrix NFUSE session. With Citrix NFuse, the client is installed, but your interface is a browser instead of a client interface. To record NFUSE sessions, you must perform a multi-protocol recording for Citrix and Web Vusers. (See Chapter 5, "Recording with VuGen.") In multi-protocol mode, VuGen generates functions from both Citrix and Web protocols during recording.

In the following example, **ctrx_mouse_click** simulates a mouse click on the left button.

```
ctrx_mouse_click(44, 318, LEFT_BUTTON, 0, CTRX_LAST);
```

For more information about the syntax and parameters, see the *Online Function Reference* (**Help** > **Function Reference**).

You can view and edit the recorded script from VuGen's main window. The API calls that were recorded during the session are displayed in VuGen, allowing you to track your actions.

Getting Started with Citrix Vuser Scripts

This section provides an overview of the process of developing Citrix ICA Vuser scripts using VuGen. In addition, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 538.

To develop a Citrix ICA script:

1 Make sure that your client and server are configured properly.

For general information about these settings, see "Setting Up the Client and Server" on page 522.

2 Record the actions using VuGen.

Invoke VuGen and create a new Vuser script. Insert bitmap and text synchronization during recording as described in "Synchronizing Replay" on page 528.

For general information about recording, see Chapter 5, "Recording with VuGen."

3 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

5 Configure the Citrix display options.

Configure the display options for replaying Citrix Vusers. These options let you show the Citrix client during replay and open a snapshot when an error occurs. For details, see "Setting the Citrix Display Settings" on page 526.

6 Configure the Run-Time settings.

The Run-Time settings control Vuser behavior during script execution. These settings include pacing, logging, think time, and connection information.

For details about the Citrix specific Run-Time settings, see "Citrix Run-Time Settings" on page 1311. For information about general Run-Time settings, see Chapter 79, "Configuring Run-Time Settings."

7 Save and run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly. 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.

For details about running the Vuser script as a standalone test, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 538 and Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Setting Up the Client and Server

Before creating a script, make sure you have a supported Citrix client installed on your machine, and that your server is properly configured. This section describes:

- Client Versions
- Server Configuration

Client Versions

In order to run your script, you must install a Citrix client on each Load Generator machine. If you do not have a client installed, you can download one from the Citrix Website www.citrix.com under the **download** section.

VuGen supports all Citrix clients with the exception of versions 8.00, version 6.30.1060 and earlier, and Citrix Web clients.

Server Configuration

To record in VuGen, you need to configure the Citrix server in the following areas:

Installing the MetaFrame Sever

Make sure the MetaFrame server (3, 4, or 4.5) is installed. To check the version of the server, select **Citrix Connection Configuration** on the server's console toolbar and select **Help** > **About**.

Configuring the MetaFrame Server to Close Sessions

Configure the Citrix server to completely close a session. After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time that client opens a new connection. Consequently, a new connection by the same client will face the same workspace from which it disconnected previously. It is preferable to allow each new test run to use a clean workspace.

The make sure that you have a clean workspace for each test, you must configure the Citrix server not to save the previous session. Instead, it should reset the connection by disconnecting from the client each time the client times-out or breaks the connection.

To configure the server:

1. Open the Citrix Connection Configuration dialog box. Select **Programs** > **Citrix** > **Administration Tools** > **Citrix Connection Configuration Tool**.
2. Select the ica-tcp connection name and select **Connection** > **Edit**. Alternatively, double-click on the connection. The Edit Connection dialog box opens.
3. Click the **Advanced** button. The Advanced Connection Settings dialog box opens.
4. In the bottom section of the dialog box, clear the **inherit user config** check box adjacent to the **On a broken or timed-out connection** list box. Change the entry for this list box to **reset**.

5 Click **OK**.

Recording Tips

When recording a script, be sure to follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. If you plan to record a simple Citrix ICA session, use a single protocol script. When recording an NFUSE Web Access session, however, you must create a multi-protocol script for Citrix ICA and Web(HTML/HTTP), to enable the recording of both protocols. For more information, see Chapter 5, "Recording with VuGen."

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Vuser Script Sections" on page 92.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start** > **Programs** > **Microsoft Word**, be sure to click on the word **Programs**.

Do not Resize Windows

Although VuGen supports the resizing of windows during recording the session, we recommend that you do not move or resize them while recording. To change the size or position of a window, double-click on the relevant **Sync on Window** step in the script's Tree view and modify the window's coordinates.

Make Sure Resolution Settings are Consistent

To insure successful bitmap synchronization, make sure that the resolution settings match. On the recording machine, check the settings of the ICA client, the Recording Options, and the Run Time settings. On the load generators, check the settings of the ICA client, and make sure that they are consistent between all load generators and recording machines. If there is an inconsistency between the resolutions, the server traffic increases in order to make the necessary adjustments.

Add Manual Synchronization Points

While waiting for an event during recording, such as the opening of an application, we recommend that you add manual synchronization points, such as **Sync on Bitmap** or **Sync on Text**. For details, see "Synchronizing Replay" on page 528.

Disable Client Updates

Disable client updates when prompted by the Citrix client. This will prevent forward compatibility issues between VuGen and newer Citrix clients that were not yet tested.

Windows Style

For **Sync on Bitmap** steps, record windows in the "classic" windows style—not the XP style.

To change the Windows style to "classic":

1. Click in the desktop area.
2. Select **Properties** from the right-click menu.
3. Select the Theme tab.
4. Select **Windows Classic** from the Theme drop down list.

5 Click **OK**.

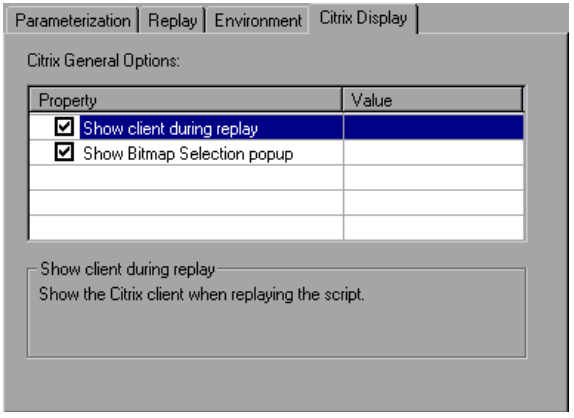
Setting the Citrix Display Settings

Before running your Citrix Vuser script, you can set several display options to be used during replay. Although these options increase the load upon the server, they are useful for debugging and analyzing your session.

To set the Citrix display options:

1 Open the General Options dialog box. Select **Tools** > **General Options** in the main VuGen window.

2 Select the **Citrix Display** tab.



3 Select **Show client during replay** to display the Citrix client when replaying the Vuser script.

4 Select **Show Bitmap Selection popup** to issue a popup message when you begin to work interactively within a snapshot. VuGen issues this message when you select the right-click menu option **Insert Sync Bitmap** or **Insert Get Text**, before you select the bitmap or text.

5 Click **OK**.

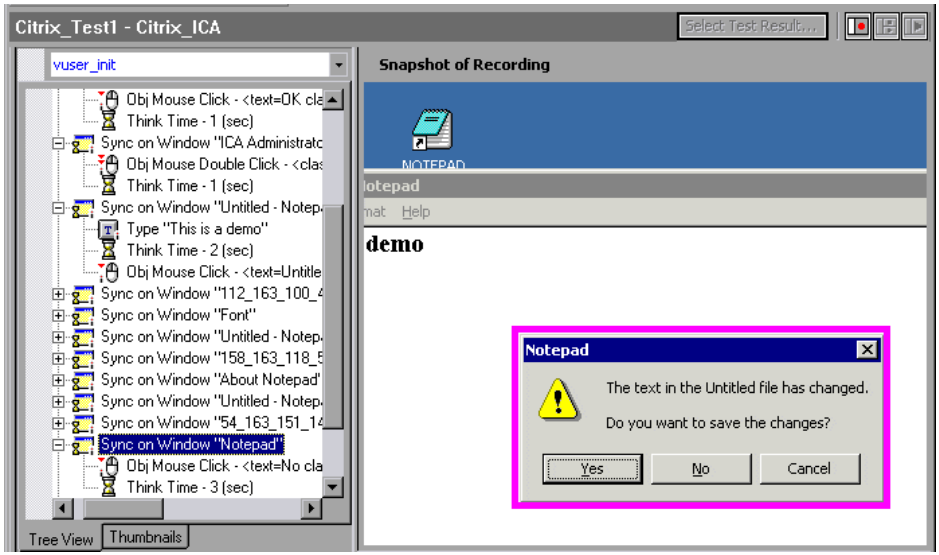
Viewing and Modifying Citrix Vuser Scripts

You can view the contents of your Vuser script in VuGen's Script view or Tree view. For general information about viewing a script, see Chapter 2, "Introducing VuGen."

In Tree view, you can view a Citrix Vuser's snapshots. Each step has an associated snapshot. In addition to displaying the client window, the snapshot also highlights the object upon which the action was performed.

- For the **Mouse** steps, a small pink square indicates where the user clicked.
- For **Sync on Bitmap**, a pink box encloses the bitmap area.

For **Sync on Window**, a pink box encloses the entire window. In the following example, the snapshot shows the **Sync On Window** step. Notepad's confirmation box is enclosed by a box indicating the exact window on which the operation was performed.



Note that VuGen saves snapshots as bitmap files in the script's data/snapshots directory. You can determine the name of the snapshot file by checking the function's arguments.

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

After recording, you can manually add steps to the script in either Script view or Tree view. For information about the various script views, see "Viewing and Modifying Vuser Scripts" on page 48.

In addition to manually adding new functions, you can add new steps interactively for Citrix Vusers, directly from the snapshot. Using the right-click menu, you can add bitmap synchronizations. When the Citrix Agent is installed on the Citrix server machine, you can also add text and object synchronizations from the right-click menu. For more information, see Chapter 30, "Citrix - Agent for Citrix Presentation Server."

To insert a function interactively:

1. Click on a step within Tree view. Make sure that a snapshot is visible.
2. Right-click and select one of the commands. A dialog box opens with the available properties.
3. Modify the desired properties and click **OK**. VuGen inserts the step into your script.

Synchronizing Replay

When running a script, it is often necessary to synchronize the actions to insure a successful replay. Synchronization refers to the timing of events within your script, waiting for windows and objects to become available before executing an action. For example, you may want to check whether a certain window has opened before attempting to press a button within the window.

VuGen automatically generates functions that synchronize the actions during replay. In addition, you can add manual synchronization functions.

Automatic Synchronization

During recording, VuGen automatically generates steps that help synchronize the Vuser's replay of the script:

- Sync on Window
- Sync on Obj Info
- Sync on Text

Sync on Window

The **Sync On Window** step instructs the Vuser to wait for a specific event before resuming replay. The available events are **Create** or **Active**. The Create event waits until the window is created. The Active event waits until the window is created and then activated (in focus). Usually VuGen generates a function with a CREATE event. If, however, the next instruction is a keyboard event, VuGen generates a function with an ACTIVE event.

In Script view, the corresponding function call to the **Sync On Window** step is **ctrx_sync_on_window**.

Sync on Obj Info

The **Sync On Obj Info** step instructs the Vuser to wait for a specific object property before resuming replay. The available attributes are **Enabled**, **Visible**, **Focused**, **Text**, **Checked**, **Lines**, or **Item**. The Enabled, Visible, Focused, and Checked attributes are boolean values that can receive the values **true** or **false**. The other attributes require a textual or numerical object value.

A primary objective of this step is to wait for an object to be in focus before performing an action upon it.

VuGen automatically generates Sync On Obj Info steps when the Citrix agent is installed and the Use Citrix Agent Input in Code Generation option is enabled in the Recording options. By default, this Recording option is enabled. For more information, see "Code Generation Recording Options" on page 1133.

```
ctrx_sync_on_obj_info("Run=snapshot9", 120, 144, TEXT, "OK",
                                        CTRX_LAST);
```

Sync on Text

The Text Synchronization step, **Sync On Text**, instructs the Vuser to wait for a text string to appear at the specified position before continuing. When replaying **Sync On Text**, Vusers search for the text in the rectangle whose modifiable coordinates are specified in the step's properties.

With an agent installation (see Chapter 30, "Citrix - Agent for Citrix Presentation Server"), you can instruct VuGen to automatically generate a text synchronization step before each mouse click or double-click. By default, automatic text synchronization is disabled. For information on how to enable this Recording option, see "Code Generation Recording Options" on page 1133.

Note, that even if you record a script with the option disabled, if you enable the option and regenerate the script, VuGen will insert text synchronization calls throughout the entire script. For more information, see "Code Generation Recording Options" on page 1133.

You can manually add text synchronizations for individual steps both during and after recording as described in "Manual Synchronization" on page 531.

In Script view, the corresponding function call to the **Sync On Text** step is **ctrx_sync_on_text_ex**.

The following segment shows a ctrx_sync_on_text_ex function that was recorded during a Citrix recording with the HP Citrix Agent installed and text synchronization enabled.

```
ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,224,
"snapshot1", CTRX_LAST);
ctrx_sync_on_text_ex (196, 198, 44, 14, "OK", "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);
ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198, LEFT_BUTTON, 0, "ICA
Seamless Host Agent=snapshot2", CTRX_LAST);
```

For more information on this function, see the *Online Function Reference* (**Help** > **Function Reference**).

Manual Synchronization

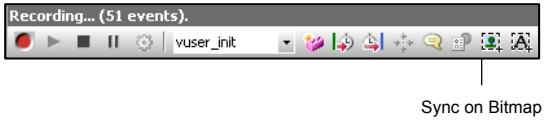
In addition to the automatic synchronization, you can manually add synchronization both during and after recording. A common use of this capability is where the actual window did not change, but an object within the window changed. Since the window did not change, VuGen did not detect or record a **Sync on Window.**

For example, if you want the replay to wait for a specific graphic image in a browser window, you insert manual synchronization. Or, if you are recording a large window with several tabs, you can insert a synchronization step to wait for the new tab's content to open.

The following section describes synchronizing on a bitmap. For information on adding a **Sync on Text** manually, see "Text Retrieval" on page 549.

Manually Adding Synchronization During Recording

To add synchronization during recording, you use the floating toolbar. The **Sync On Bitmap** function lets you to mark an area within the client window that needs to be in focus before resuming replay.



To mark a bitmap area for synchronization:

1 Click the **Insert Sync on Bitmap** button on the toolbar.

2 Mark a rectangle around the desired bitmap. In Tree view, VuGen generates a Sync on Bitmap step after the current step. In Script view, VuGen generates a ctrx_sync_on_bitmap function with the selected coordinates as arguments.

```
ctrx_sync_on_bitmap(93, 227, 78, 52,
                    "66de3122a58baade89e63698d1c0d5dfa", CTRX_LAST);
```

During replay, Vusers look for the bitmap at the specified coordinates, and wait until it is available before resuming the test.

Manually Adding Synchronization After Recording

You can also add synchronization after the recording session. To add a synchronization step, right-click in the snapshot window and select a synchronization option:

- **Sync on Bitmap.** Waits until a bitmap appears
- **Sync on Obj Info.** Waits until an object's attributes have the specified values (agent installations only)
- **Sync on Text.** Waits until the specified text is displayed (agent installations only)

During recording, the bitmaps generated for the **Sync on Bitmap** step are saved under the script's **data/snapshots** directory. If synchronization fails during replay, VuGen generates a new bitmap that you can examine to determine why synchronization failed. VuGen displays both bitmaps in the Failed Bitmap Synchronization dialog box. For more information, see "Failed Bitmap Synchronization" on page 535.

The bitmap name has the format of sync_bitmap_<hash_value>.bmp. It is stored in the script's output directory, or for a scenario or profile, wherever the output files are written.

Additional Synchronizations

In addition, you can add several other steps that affect the synchronization indirectly:

- Setting the Waiting Time
- Checking if a Window Exists or Closed
- Waiting for a Bitmap Change

Setting the Waiting Time

The **Set Waiting Time** step sets a waiting time for the other Citrix synchronization functions. This setting applies to all functions that follow it within the script. For example, if your **Sync on Window** steps are timing out, you can increase the default timeout of 60 seconds to 180.

To insert this step, select **Insert** > **Add Step** > **Set Waiting Time**.

Checking if a Window Exists or Closed

The Win Exist step checks if a window is visible in the Citrix client. By adding control flow statements, you can use this function to check for a window that does not always open, such as a warning dialog box. In the following example, ctrx_win_exist checks whether a browser was launched. The second argument indicates how long to wait for the browser window to open. If it did not open in the specified time, it double-clicks its icon.

```
if (!ctrx_win_exist("Welcome",6, CTRX_LAST))
  ctrx_mouse_double_click(34, 325, LEFT_BUTTON, 0, CTRX_LAST)
```

To insert this step, select **Insert** > **Add Step** > **Win Exist**.

Another useful application for this step is to check if a window has been closed. If you need to wait for a window to close, you should use a synchronization step such as **UnSet Window** or **ctrx_unset_window**.

For detailed information about these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Waiting for a Bitmap Change

In certain cases, you do not know what data or image will be displayed in an area, but you do expect it to change. To emulate this, you can use the **Sync on Bitmap Change** step or its corresponding function, **ctrx_sync_on_bitmap_change**. Perform a right-click in the snapshot, and select an **Insert Sync on Bitmap** from the right-click menu. VuGen inserts the step or function at the location of the cursor.

The syntax of the functions is as follows:

```
ctrx_sync_on_bitmap (x_start, y_start, width, height, hash, CTRX_LAST);
ctrx_sync_on_bitmap_change (x_start, y_start, width, height,
                            [initial_wait_time,] [timeout,]
                            [initial_bitmap_value,] CTRX_LAST);
```

The following optional arguments are available for **ctrx_sync_on_bitmap_change**:

- initial wait time value—when to begin checking for a change.
- a timeout—the amount of time in seconds to wait for a change to occur before failing.
- initial bitmap value—the initial hash value of the bitmap. Vusers wait until the hash value is different from the specified initial bitmap value.

In the following example, the recorded function was modified and assigned an initial waiting time of 300 seconds and a timeout of 400 seconds.

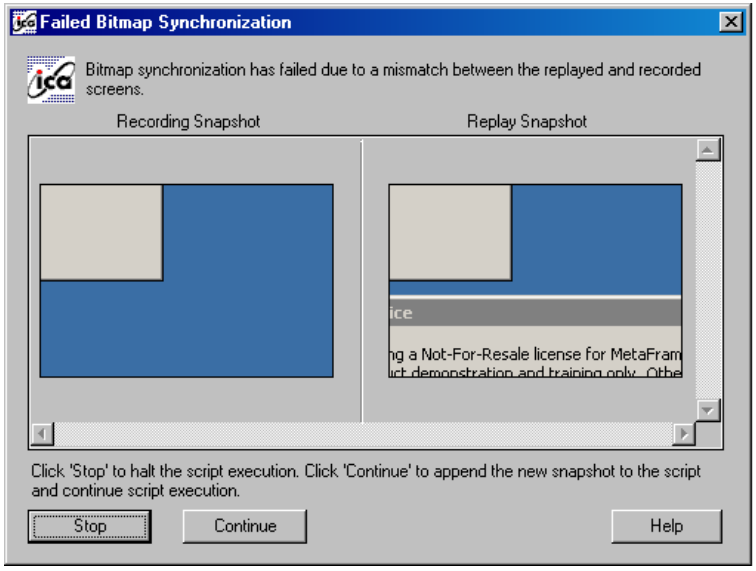
```
ctrx_sync_on_bitmap_change(93, 227, 78, 52,
                    300,400, "66de3122a58baade89e63698d1c0d5dfa",CTRX_LAST);
```

Note: If you are using **Sync on Bitmap**, make sure that the Configuration settings in the Controller, Load Generator machine, and screen are the same. Otherwise, VuGen may be unable to find the correct bitmaps during replay. For information on how to configure the client settings, see "Configuration Recording Options" on page 1130.

Failed Bitmap Synchronization

The Failed Bitmap Synchronization dialog box opens when there is a mismatch between the Recording and Replay snapshots during script replay.

You can indicate whether or not you want to mark the mismatch as an error or adopt the changes.



When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution. You can specify Continue on Error for a specific function as described in "Continuing on Error" on page 541.
- **Continue.** Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.

Understanding ICA Files

Citrix ICA client files are text files that contain configuration information for the applications accessed through the Citrix client. These files must have an .ica extension and must conform to the following format:

```
[WFClient]
Version=
TcpBrowserAddress=

[ApplicationServers]
AppName1=

[AppName1]
Address=
InitialProgram=#
ClientAudio=
AudioBandwidthLimit=
Compress=
DesiredHRES=
DesiredVRES=
DesiredColor=
TransportDriver=
WinStationDriver=

Username=
Domain=
ClearPassword=
```

Note: When you load an ICA file using the Recording Options, VuGen saves the file together with your script, eliminating the need to copy the ICA file to each load generator machine.

The following example shows a sample ICA file for using Microsoft Word on a remote machine through the Citrix client:

```
[WFClient]
Version=2
TcpBrowserAddress=235.119.93.56

[ApplicationServers]
Word=

[Word]
Address=Word
InitialProgram=#Word
ClientAudio=On
AudioBandwidthLimit=2
Compress=On
DesiredHRES=800
DesiredVRES=600
DesiredColor=2
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

Username=test
Domain=user_lab
ClearPassword=test
```

For more information, see the Citrix Website www.citrix.com.

Using Citrix Functions

During a Citrix recording session, VuGen generates functions that emulate the communication between a client and a remote server. The generated functions have a **ctrx** prefix. You can also manually edit any of the functions into your Vuser script after the recording session. For example, **ctrx_obj_mouse_click** emulates a mouse click for a specific object.

For more information about the **ctrx** functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Note that for the functions that specify a window name, you can use the wildcard symbol, an asterisk (*). You can place the wildcard at the beginning, middle, or end of the string.

Tips for Replaying and Troubleshooting Citrix Vuser Scripts

The following sections provide guidelines and tips for Citrix Vusers in the following areas:

- Replay Tips
- Debugging Tips

For recording tips, see "Recording Tips" on page 524.

Replay Tips

Wildcards

You can use wildcards (*) in defining window names. This is especially useful where the window name may change during replay, by its suffix or prefix.

In the following example, the title of the **Microsoft Internet Explorer** window was modified with a wildcard.

```
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0,
"Welcome to MSN.com - Microsoft Internet Explorer");
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0,
"* - Microsoft Internet Explorer");
```

For more information, see the Function Reference (**Help > Function Reference**).

Set Initialization Quota

To prevent overloading by multiple Vusers while connecting, set an initialization quota of 4 to 10 Vusers (depending on the capacity of the server) or apply ramp-up initialization using the Scheduler.

Enable Think Time

For best results, do not disable think time in the Run-Time settings. Think time is especially relevant before the **ctrx_sync_on_window** and **ctrx_sync_on_bitmap** functions, which require time to stabilize.

Regenerate Script

During recording, VuGen saves all of the agent information together with the script. By default, it also includes this information in the script, excluding the **Sync On Text** steps. If you encounter text synchronization issues, then you can regenerate the script to include the text synchronization steps.

In addition, if you disabled the generation of agent information in the Recording options, you can regenerate the script to include them.

Regenerating scripts is also useful for scripts that you manually modified. When you regenerate the script, VuGen discards all of your manual changes and reverts back to the originally recorded version.

To regenerate a script, select **Tools** > **Regenerate** and select the desired options. For more information about regenerating scripts, see "Regenerating a Vuser Script" on page 113.

Set Consistency Between Machines

If you intend to replay the script on another machine, make sure that the following items are consistent between the record and replay machines: Window Size (resolution), Window Colors, System Font and the other Default Options settings for the Citrix client. These settings affect the hash value of bitmaps, and inconsistencies may cause replay to fail. To view the Citrix Client settings, select an item from the Citrix program group and select **Application Set Settings** or **Custom Connection Settings** from the right-click menu. Select the Default Options tab.

Increasing the Number of Vusers per Load Generator Machine

Load Generator machines running Citrix Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine, also known as the GDI (Graphics Device Interface). To increase the number of Vusers per machine, you can open a terminal server session on the machine which acts as an additional load generator.

The GDI count is Operating System dependent. The actual GDI (Graphics Device Interface) count for a heavily loaded machine using LoadRunner is approximately 7,500. The maximum available GDI on Windows 2000 machines is 16,384.

For more information on creating a terminal server session, see the Terminal Services topics in the HP LoadRunner Controller.

Note: By default, sessions on a terminal server use a 256-color set. If you intend to use a terminal session for load testing, make sure to record on machines with a 256-color set.

Debugging Tips

Single Client Installation

If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine. To verify that only one client is installed, open the Add/Remove Programs dialog box from the Control Panel and make sure that there is only one entry for the Citrix ICA client.

Add Breakpoints

Add breakpoints to your script in VuGen to help you determine the problematic lines of code.

Synchronize Your Script

If replay fails, you may need to insert synchronization functions into your script to allow more time for the desired windows to come into focus. Although you can manually add a delay using **lr_think_time**, we recommend that you use one of the synchronization functions discussed in "Synchronizing Replay" on page 528.

Continuing on Error

You can instruct Vusers to continue running even after encountering an error, such as not locating a matching window. You specify Continue on Error for individual steps.

This is especially useful where you know that one of two windows may open, but you are unsure of which. Both windows are legal, but only one will open.

To indicate Continue on Error:

In **Tree view**, right-click on the step and select **Properties**. In the **Continue on Error** box, select the **CONTINUE_ON_ERROR** option.

In **Script view**, locate the function and add **CONTINUE_ON_ERROR** as a final argument, before **CTRX_LAST**.

This option is not available for the following functions: **ctrx_key**, **ctrx_key_down**, **ctrx_key_up**, **ctrx_type**, **ctrx_set_waiting_time**, **ctrx_save_bitmap**, **ctrx_execute_on_window**, and **ctrx_set_exception**.

Extended Log

You can view additional replay information in the Extended log. To do this, enable Extended logging in the Run-Time settings (F4 Shortcut key) **Log** tab. You can view this information in the Replay Log tab or in the output.txt file in the script's directory.

Snapshot Bitmap

When an error occurs, VuGen saves a snapshot of the screen to the script's **output** directory. You can view the bitmap to try to determine why the error occurred.

During recording, the bitmaps generated for the **ctrx_sync_on_bitmap** function are saved under the script's **data** directory. The bitmap name has the format of hash_value.bmp. If synchronization fails during replay, the generated bitmap is written to the script's output directory, or if you are running it in a scenario, to wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.

Show Vusers

To show Vusers during a scenario, enter the following in the Vuser command line box: -lr_citrix_vuser_view. In the Controller, open the Vuser Details dialog box and click **More** to expand the dialog box. Note that this will affect the scalability of the test, so this should only be done to examine a problematic Vuser's behavior.

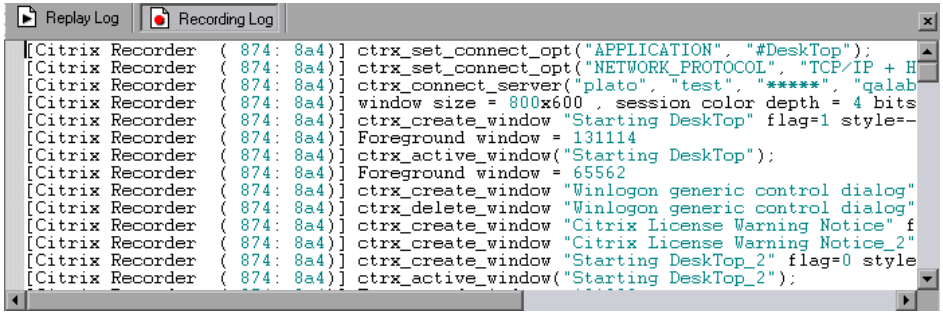
To reduce the effect on the script's scalability, you can show the details for an individual Vuser by adding the Vuser's ID at the end of the command line: -lr_citrix_vuser_view <*VuserID*>.

To open multiple Vusers, place a comma-separated list of IDs after the command line. Do not use spaces, but you may use commas or dashes. For example, 1,3-5,7 would show Vusers 1,3,4,5, and 7, but would not show Vuser 2, 6 or any Vuser with an ID higher than 7.

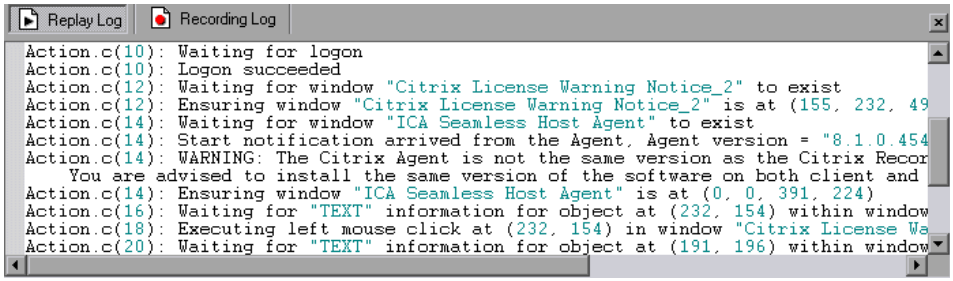
View Recording and Replay Logs

To see detailed information about the recording, view the Recording and Replay logs in the Output window. To open the Output window, select **View > Output Window**.

To view the Recording Log, select the **Recording Log** tab. VuGen displays a detailed log of all functions that were generated by the recording and the warning messages and errors that were issued during that time.



To view the Replay Log, select the **Replay Log** tab. VuGen provides a description of all actions performed by VuGen and any warnings and errors that were issued during the replay.



To go directly to the step in the script associated with the log message, double-click on the message in the Replay log.

Note that the extent of information in the Replay log depends on the Log Run-Time settings. For more information, see Chapter 79, "Configuring Run-Time Settings."

30

Citrix - Agent for Citrix Presentation Server

The Agent for Citrix Presentation Server is an optional utility that you can install on the Citrix server. It provides you with several important benefits that can enhance your script:

This chapter includes:

About the Agent for Citrix Presentation Server

The Agent for Citrix Presentation Server is an optional utility that you can install on the Citrix server. It provides enhancements to the normal Citrix functionality. It is provided in the product's installation disk and you can install it on any Citrix server.

The agent provides you with the following benefits:

- Intuitive and readable scripts
- Built-in synchronization
- Detailed Information about relevant objects

➤ Ability to work interactively within the Client window

Using the Agent for Citrix Presentation Server Features

The Agent for Citrix Presentation Server provides enhancements to the normal Citrix functionality. The following sections describe these enhancements and provide details and sample scripts.

Object Detail Recording

When the Agent for Citrix Presentation Server is installed, VuGen records specific information about the active object instead of general information about the action. For example, VuGen generates **Obj Mouse Click** and **Obj Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows the same mouse-click action recorded with and without the agent installation. Note that with an agent, VuGen generates ctrx_obj_xxx functions for all of the mouse actions, such as click, double-click and release.

```
/* WIthout Agent Installation */
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0, test3.txt - Notepad);

/* WIth Agent Installation */
ctrx_obj_mouse_click("<text=test3.txt - Notepad class=Notepad>" 573,
        61, LEFT_BUTTON, 0, test3.txt - Notepad=snapshot21, CTRX_LAST);
```

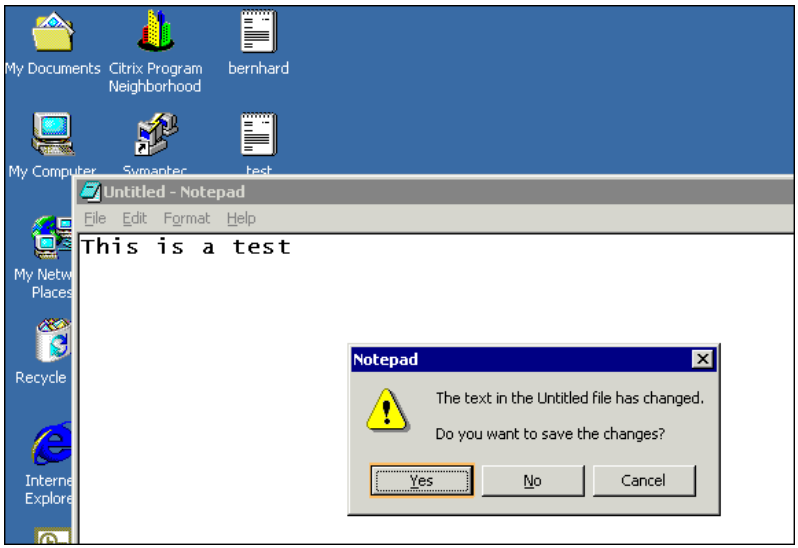
In the example above, the first argument of the **ctrx_obj_mouse_click** function contains the text of the window's title and the class, Notepad. Note that although the agent provides additional information about each object, Vusers only access objects by their window name and its coordinates.

Active Object Recognition

The agent installation lets you see which objects in the client window are detected by VuGen. This includes all Windows Basic Objects such as edit boxes, buttons, and item lists in the current window.

To see which objects were detected, you move your mouse through the snapshot. VuGen highlights the borders of the detected objects as the mouse passes over them.

In the following example, the **Yes** button is one of the detected objects.



Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When no agent is installed, you are limited to the **Insert Mouse Click**, **Insert Mouse Double Click**, **Insert Sync on Bitmap** and **Insert Get Bitmap Value**. If you are using a 256-color set, the **Insert Sync on Bitmap** and **Get Bitmap Value** steps are not available from the right-click menu.

When the Agent for Citrix Presentation Server is installed, the following additional options are available from the right-click menu of window in focus:

- **Obj Get Info** and **Sync on Obj Info**. Provide information about the state of the object: ENABLED, FOCUSED, VISIBLE, TEXT, CHECKED, and LINES.

- **Insert Sync on Obj Info**. Instructs VuGen to wait for a certain state before continuing. This is generated as a **ctrx_sync_on_obj_info** function.
- **Insert Obj Get Info**. Retrieves the current state of any object property. This is generated as a **ctrx_get_obj_info** function.
- **Insert Sync on Text** and **Get Text**. These are discussed in the section "Text Retrieval" on page 549.

These commands are interactive—when you insert them into the script, you mark the object or text area in the snapshot.

In the following example, the ctrx_sync_on_obj_info function provides synchronization by waiting for the Font dialog box to come into focus.

```
ctrx_sync_on_obj_info("Font", 31, 59, FOCUSED, "TRUE", CTRX_LAST);
```

Utilizing VuGen's ability to detect objects, you can perform actions on specific objects interactively, from within the snapshot.

To insert a function interactively using the agent capabilities:

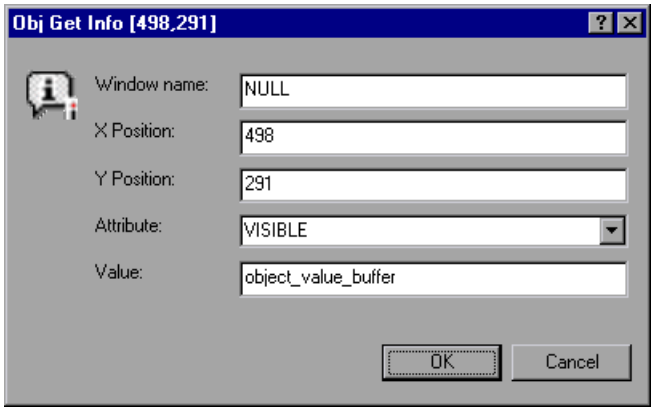
1. Click at a point within the tree view to insert the new step. Make sure that a snapshot is visible.
2. Click within the snapshot.
3. To mark a bitmap, right-click on it and select **Insert Sync on Bitmap**.

   VuGen issues a message indicating that you need to mark the desired area by dragging the cursor. Click **OK** and drag the cursor diagonally across the bitmap that you want to select.

   When you release the mouse, VuGen inserts the step into the script after the currently selected step.

4 For all other steps, move your mouse over snapshot objects to determine which items are active—VuGen highlights the borders of active objects as the mouse passes over them.

Right-click and select one of the Insert commands. A dialog box opens with the step's properties.



Set the desired properties and click **OK**. VuGen inserts the step into your script.

Text Retrieval

With the agent installed, VuGen lets you save standard text to a buffer. Note that VuGen can only save true text—-not a graphical representation of text in the form of an image.

You save the text using the **Sync On Text** step either during or after recording.

To retrieve a text string:

1 During recording: Click the **Insert Sync On Text** button on the toolbar.

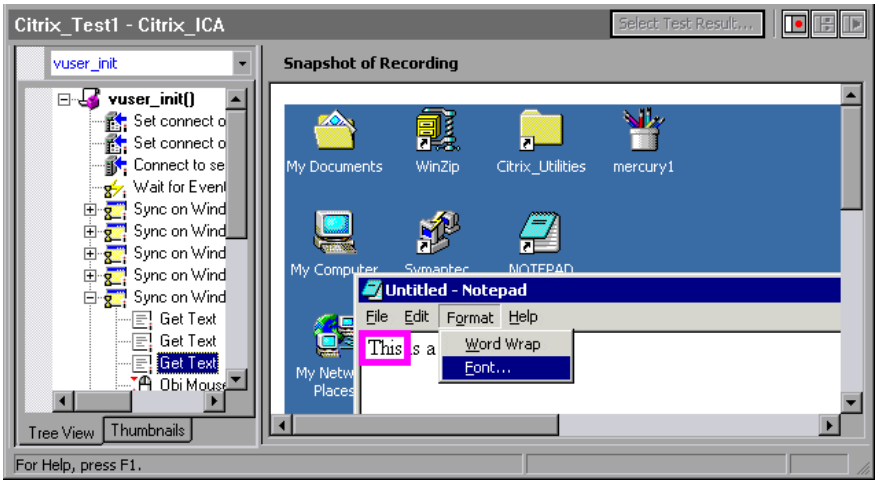
After recording: Select **Insert Sync On Text** from the snapshot's right-click menu. The Bitmap Selection dialog opens, indicating that you are inserting a synchronization or informational function and that you need to mark an area.

2 Click at the corner of the text that you want to capture, drag the mouse diagonally to mark the text you want to save, and release the mouse button.

3 If you add the step during recording, VuGen places a **Insert Sync On Text** step at the current location and saves the text to a buffer.

If your are adding the step after recording, VuGen prompts you with the Sync On Text Ex dialog box, allowing you to manually specify text.

VuGen marks the saved text with a pink box. In the following snapshot, the **Insert Sync On Text** step retrieved the text **This**.



Data Execution Prevention (DEP) and Citrix Performance

DEP is a security feature included in Windows version XP Service Pack 2 and later. DEP can interfere with some functions of the Agent for Citrix Presentation Server. We recommend that the users disable DEP to ensure that the agent runs properly.

To disable DEP

1 From the desktop, right click **My Computer** and select **Properties**. Alternatively, click **Start** > **Control Panel** > **System**. The **System Properties** dialog box opens.

2 Click the **Advanced** tab.

3 In the **Startup and Recovery** pane, click **Settings**.

4 Under **System startup**, click **Edit**.

5 Locate the **/noexecute** element and change the value to **AlwaysOff**.

6 Save the file and reboot your machine.

Installing the Citrix Presentation Server Agent

The installation file for the Agent for Citrix Presentation Server is located on the LoadRunner installation disk, under the **Additional Components\Agent for Citrix Presentation Server** folder.

Note that the agent should only be installed on your Citrix server machine—not Load Generator machines.

If you are upgrading the agent, make sure to uninstall the previous version before installing the next one (see uninstallation instructions below).

To install the Agent for Citrix Presentation Server:

1. If your server requires administrator permissions to install software, log in as an administrator to the server.
2. If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

3. Locate the installation file, **Setup.exe**, on the product installation disk in the **Additional Components\Agent for Citrix Presentation Server\Win32 or Win64** directory.
4. Follow the installation wizard to completion.

Note: After installation the agent will only be active for LoadRunner invoked Citrix sessions—it will not be active for users who start a Citrix session without LoadRunner.

To disable the agent, you must uninstall it.

To uninstall the Agent for Citrix Presentation Server:

1. If your server requires administrator privileges to remove software, log in as an administrator to the server.

2 Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Citrix Presentation Server 32 or 64** and click **Change/Remove**.

Effects and Memory Requirements of the Citrix Agent

When you run Citrix Vusers with the agent installed, each Vuser runs its own process of **ctrxagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine (about 7%).

The memory requirements per Citrix ICA Vuser (each Vuser runs its own **ctrxagent.exe** process) is approximately 4.35 MB. To run 25 Vusers, you would need 110 MBs of memory.

Sample Script

The following script illustrates a true Citrix ICA session with an agent.

```
vuser_init ()
{
   ctrx_set_connect_opt (NETWORK_PROTOCOL, "TCP/IP + HTTP");
   ctrx_connect_server ("Plato", "test", lr_decrypt("428c4445a14409b9"), "QAlab");
   ctrx_wait_for_event ("LOGON");
   ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,224,
"snapshot1", CTRX_LAST);
   ctrx_sync_on_text (196, 198, "OK", TEXT, "ICA Seamless Host Agent=snapshot2",
CTRX_LAST);
   ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198, LEFT_BUTTON, 0, "ICA
Seamless Host Agent=snapshot2", CTRX_LAST);
   lr_think_time (73);
return 0;
}
```

31

Remote Desktop Protocol (RDP)

VuGen allows you to record the actions of a client communicating with its server using the Microsoft Remote Desktop Protocol (RDP). The resulting script is called an RDP Vuser script.

This chapter includes:

About Microsoft Remote Desktop Protocol (RDP) Vuser Scripts

The Microsoft Remote Desktop Protocol (RDP) allows users to connect to a remote computer. For example, you can use RDP to connect to a central and powerful server for working on specific business applications or graphic terminals. This provides the user with the same look and feel as if they are working on a standalone PC.

Note: RDP versions 5.1 and later have an **Experience** tab that allows you to set various options. This tab is not supported by VuGen recording. All options are set to the ON position.

Recording Tips

When recording a script, be sure to follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. For example, to record both RDP traffic and Web responses, create a multi-protocol script for RDP and Web to enable the recording of both protocols. For more information, see Chapter 5, "Recording with VuGen."

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Vuser Script Sections" on page 92.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

You should also configure your terminal server to end disconnected sessions. Select **Administrative Tools** > **Terminal Services Configuration** > **Connection Properties** > **Sessions** > **Override User Settings** and set the server to end disconnected sessions.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start** > **Programs** > **Microsoft Word**, be sure to click on the word **Programs**.

Recording an RDP Vuser Script

Before recording, you set the desired recording options as described above.

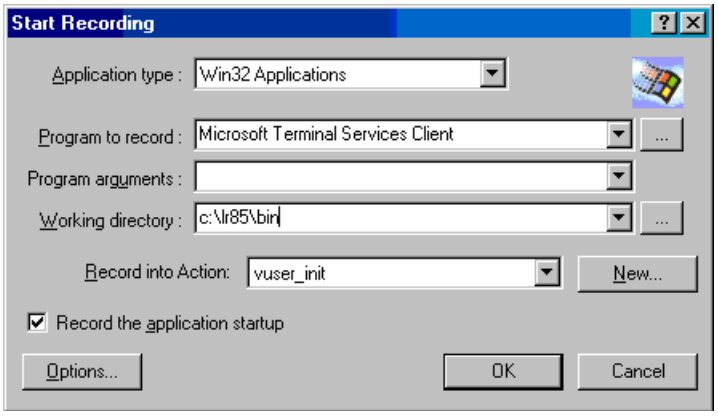
Note: Whenever recording an RDP script, make sure that you record the act of logging off the server. This makes sure that when you run the script you will begin with a new session.

At any point during recording, you can mark an area of the remote desktop to synchronize upon. The image is stored on disk as **snapshot_xxx.png** where **xxx** is a sequential index that starts at 1 and is increased by 1 for each image and includes images from other steps, for example, mouse clicks. A matching function appears in the script at this point during code generation.

To create an RDP Vuser script:

1 Select **File** > **New** or click the **New** button. The New Virtual User dialog box opens.

2 Select **Microsoft Remote Desktop Control Protocol (RDP)**. The Start Recording dialog box opens.



3 Click **Options** to set the Recording Options. For more information, see "RDP Recording Options" on page 1125.

4 Select the **RDP: Login** node. Select one of the session options: **Run Client**, **Connection File**, or **Default Connection File**.

5 Select the **RDP: Code Generation** node and enable the desired options.

6 During recording to select a screen region for synchronization, click the **Sync on image button** on the recording toolbar and indicate an area for synchronization.

Note: You can also add an image synchronization after recording the script. Right-click on the image snapshot and select **Insert Sync On Image** from the menu.

7 Stop recording and save the script.

Running RDP Vuser Scripts

Before running an RDP script, you can set the run-time settings to customize the behavior of the script. You can set general run-time settings for all protocols, such as think time, iterations, pacing, and logging, as described in Chapter 79, "Configuring Run-Time Settings."

For RDP-specific settings, see above.

To run an RDP script:

1. Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the toolbar, or select **Vuser** > **Run-Time Settings**.
2. Select the **Configuration** node. Select the desired settings.
3. Select the **Synchronization** node. Select the desired settings.
4. Click **OK** to accept the run-time settings and close the dialog box.
5. Click the Run button or select **Vuser** > **Run**.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Working with Clipboard Data

VuGen allows you to copy and paste the textual contents of a clipboard during an RDP session. You can copy them locally and paste them remotely, or vice versa—copy them from the remote machine and paste them locally. The copying of text is supported in TEXT, LOCALE, and UNICODE formats.

VuGen generates separate functions when providing or saving the clipboard data.

The following example illustrates a copy operation on a local machine and a paste on a remote machine:

```
//Notifies the Remote Desktop that new data is available in the Local machine's
//clipboard.The data can be provided in three formats: TEXT, UNICODE and LOCALE
rdp_notify_new_clipboard_data(
"StepDescription=Send local clipboard formats 1",
"Snapshot=snapshot1.inf",
"FormatsList=RDP_CF_TEXT|RDP_CF_UNICODE|RDP_CF_LOCALE",
RDP_LAST );

rdp_key(
"StepDescription=Key Press 2",
"Snapshot=snapshot_9.inf",
"KeyValue=V",
"KeyModifier=CONTROL_KEY",
RDP_LAST );

//Provides clipboard data to the Remote Desktop when it requests the data.
rdp_send_clipboard_data(
"StepDescription=Set Remote Desktop clipboard 1",
"Snapshot=snapshot1.inf",
"Timeout=20",
REQUEST_RESPONSE, "Format=RDP_CF_UNICODE", "Text=text for clipboard",
RDP_LAST);
```

This example illustrates a copy operation on a remote machine and a paste on a local machine:

```
rdp_key(
"StepDescription=Key Press 2",
"Snapshot=snapshot_9.inf",
"KeyValue=C",
"KeyModifier=CONTROL_KEY",
RDP_LAST);

// The function requests the Remote Desktop UNICODE text and saves it to a
//parameter
rdp_receive_clipboard_data(
"StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot1.inf",
"ClipboardDataFormat=RDP_CF_UNICODE",
"ParamToSaveData=MyParam",
RDP_LAST);
```

Normally, the Remote Desktop clipboard data is saved in UNICODE format. If the Remote Desktop requests data in the TEXT or LOCALE formats, the **rdp_send_clipboard_data** function automatically converts the content of MyParam from UNICODE into the requested format and sends it to the Remote Desktop. The Replay log indicates this conversions with an informational message. If the conversion is not possible, the step fails.

For more information about the rdp functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Correlating Clipboard Parameters

During a recording session, if the client sends the server the same data as it received, then VuGen replaces the sent data with a parameter during code generation. VuGen only performs this correlation when the received and sent data formats are consistent with one another.

The following example shows how the same parameter, **MyParam**, is used for both receiving and sending the data.

```
// Receive the data from the server
rdp_receive_clipboard_data("StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot_9.inf",
"Timeout=0",
"ClipboardDataFormat=RDP_CF_UNICODETEXT",
"ParamToSaveData=MyParam",
RDP_LAST);
…
// Send the data to the server
rdp_send_clipboard_data("StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot_9.inf",
"Timeout=10",
REQUEST_RESPONSE, "Format=RDP_CF_UNICODETEXT", "Text={MyParam}",
RDP_LAST);
```

Synchronizing Replay

An RDP session executes remotely. All keyboard and mouse handling is done on the server, and it is the server that reacts to them. For example, when you double-click an application on the desktop, it is the server that realizes a double-click took place and that the application must be loaded and displayed.

When an RDP client connects to a server, it does two things:

- It sends the server coordinates of actions. For example, 'clicked the left mouse button at coordinates (100, 100) on the screen'.
- It receives images from the server showing the current status of the screen after the action took place

The RDP client (and therefore, LoadRunner) does not know that the screen contains windows, buttons, icons, or other objects. It only knows the screen contains an image and at what coordinates the user performed the action. To allow the server to correctly interpret the actions, you set synchronization points within the script. These points instruct the script to wait until the screen on the server matches the stored screen before continuing.

To add image synchronization points to a script:

1. View the script in Tree view. Select **View** > **Tree view**.
2. Select an operation to which you would like to add a synchronization point.
3. Right-click on the image snapshot and select **Insert Synch On Image** from the menu. The cursor will change to a cross-hair.
4. Mark the area on the screen that you would like to synchronize upon by clicking on the left button and dragging the box to enclose the area. When you release the mouse button, the Sync on Image dialog box opens.
5. Click **OK**. VuGen adds a new Sync on Image step before the selected step. When you select this step, VuGen displays a snapshot that contains a pink box around the area you selected for synchronization.

The next time you replay the script, it will wait until the image returned by the server matches the image you selected.

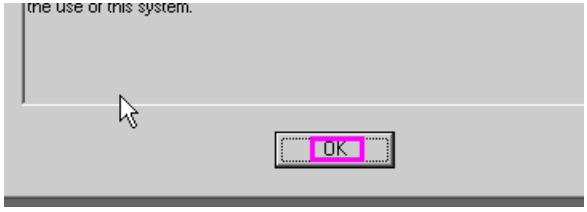
Image Synchronization Tips

Use the following guidelines for effective image synchronization:

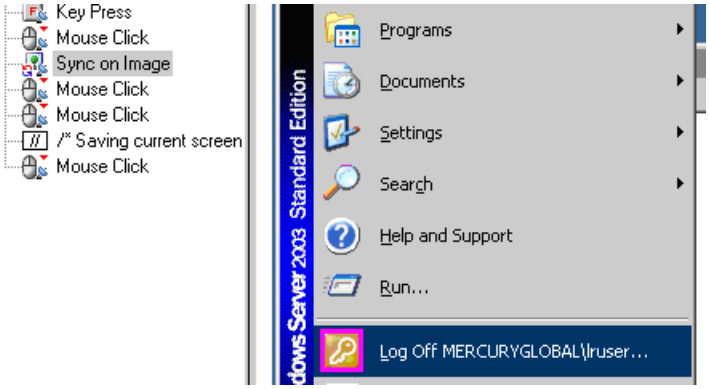
Synchronize on Smallest Significant Area

When synchronizing on an image, try to synchronize only the part of the image that is necessary. Additional details within the image may not be reproduced during replay and could result in a synchronization failure.

For example, when synchronizing on an image of a button, select only the text itself and not the dotted lines around the text as they may not appear during replay.



When synchronizing a highlighted area, try to capture only the part of the image that is not effected by the highlighting. In the following example, perform a synchronization on the Log Off icon, but not the entire button, since the highlighting may not appear during replay, and the color could vary with different color schemes.



Synchronize Before Every User Action

You need to synchronize before every mouse operation. It is also recommended that you synchronize before the first **rdp_key** / **rdp_type** operation that follows a mouse operation.

Failed Image Synchronization

The Failed Image Synchronization dialog boxes open when there is a mismatch between the Recording and Replay snapshots during script replay. There are several Failed Image Synchronization dialog boxes:

- Append Snapshot
- Raise Tolerance
- Lower Tolerance
- No Snapshot

Note: Raising or lowering the tolerance level from the dialog box only changes the level for that step. To change the tolerance level for the whole script, change the **Default Image Sync Tolerance** setting as described in "RDP Synchronization Run-Time Settings" on page 1307.

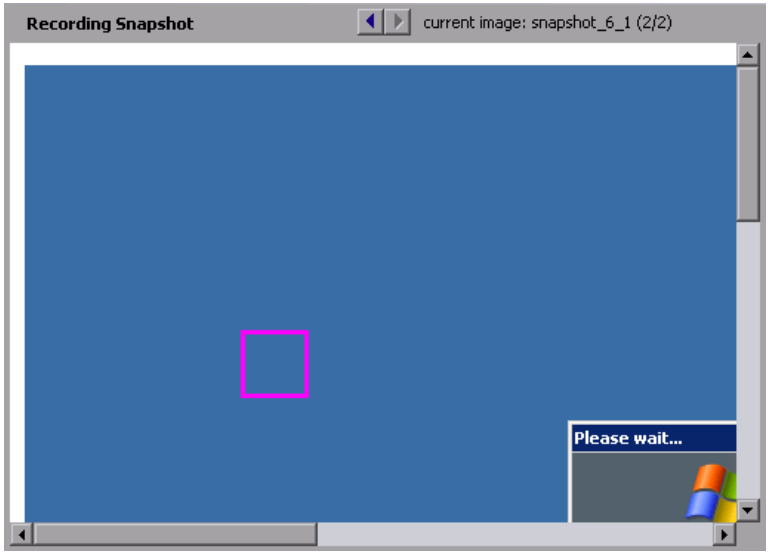
Append Snapshot

The Failed Image Synchronization - Append Snapshot dialog box opens when the replay image is so different from the recorded image that changing the tolerance level will not help. You can indicate whether or not you want to mark the mismatch as an error or adopt the changes.

When this dialog box opens, click on one of the following buttons to proceed:

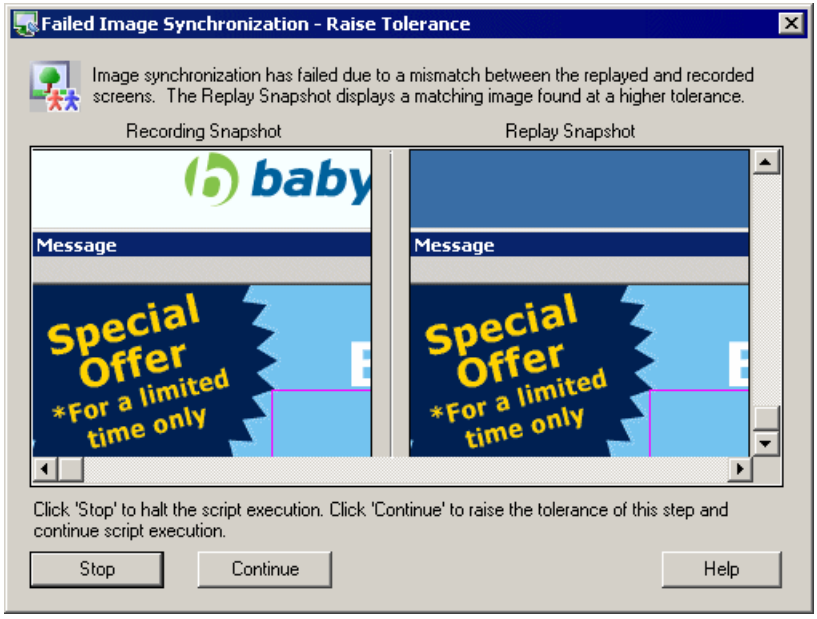
- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.

When you append a new snapshot to the original snapshot, VuGen adds it to the current step. You can then view both the original and appended snapshots by clicking the arrows above the Recording snapshot. The Replay snapshot only shows a single image, the image found during replay.



Raise Tolerance

The Failed Image Synchronization - Raise Tolerance dialog box opens when the script replay failed to find the exact image requested, but if the tolerance level for performing synchronization on images was relaxed, then it would have succeeded in finding the image.

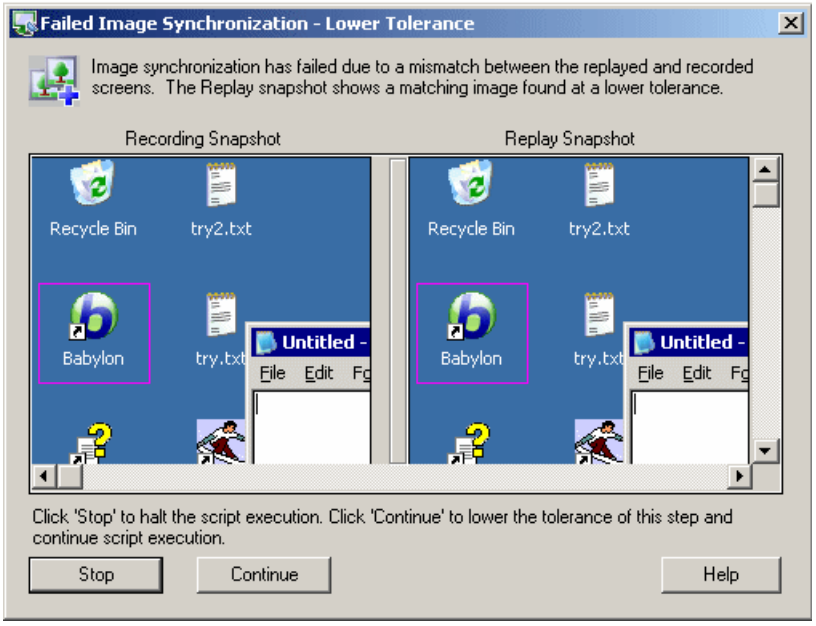


When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch and raise the tolerance level so that VuGen permits a greater degree of a mismatch between the recorded images and those displayed during the replay.

Lower Tolerance

The Failed Image Synchronization - Lower Tolerance dialog box opens when the script replay fails to meet the **NotAppear** or **Change** conditions. VuGen detected an image match where you expected it not to detect one. If the tolerance level was reduced, the recorded and replay images would not match, and the **NotAppear** or **Change** conditions would be met resulting in a successful replay.

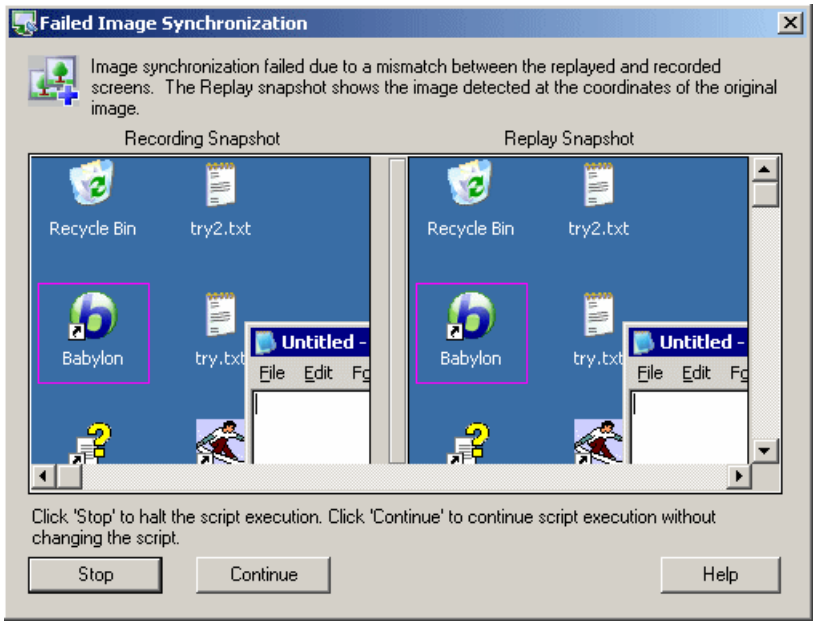


When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch and lower the tolerance level so that VuGen permits a less degree of a mismatch between the recorded images and those displayed during the replay.

No Snapshot

The Failed Image Synchronization dialog box opens when the script replay fails to meet any of the synchronization conditions such as **NotAppear** or **Change**. VuGen did not find another image at the original coordinates that could be appended to the script. You can instruct VuGen to continue replay despite the mismatch. This is ideal for situations where you know that the missing image is not essential to the business process.



When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch, and do not make any changes in the script. Continue script execution despite the mismatch.

Shifted Coordinates

When replaying a script, a recorded object may appear at different coordinates on the screen. The object is the same, but its placement has been shifted. For example, during recording a window opened at coordinates (100, 100), but during replay at (200, 250).

In this case, the synchronization point will automatically find the new coordinates without any intervention on your part. It will automatically note the difference of 100 pixels in the horizontal axis and 150 pixels in the vertical axis.

All subsequent mouse operations that are coordinate dependent will use the modified coordinates, so that a mouse click recorded at (130, 130) will be replayed to (230, 280) = (130 + 100, 130 + 150).

You control the shifting of the coordinates through the **AddOffsetToInput** parameter in the **rdp_sync_on_image** step. You can override this parameter to either add or not add the differences in location during replay to the recorded coordinates for any further operations. If you do not override this parameter, VuGen takes its value from the default setting in the run-time settings.

The corresponding parameter in the operations (for example **rdp_mouse_click** or **rdp_mouse_drag**) is **Origin**. This parameter decides whether the operation should take its coordinates only from the 'clean' values that were recorded, or whether it should take into account the differences that were added by the last synchronization point. If not explicitly specified, VuGen takes the value for this parameter from the run-time settings.

32

RDP - Agent for Microsoft Terminal Server

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides you with several important benefits that can enhance your script.

This chapter includes:

About the Agent for Microsoft Terminal Server

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides enhancements to the normal RDP functionality. It is provided in the product's DVD and you can install it on any RDP server.

The agent provides you with the following benefits:

- Intuitive and readable scripts
- Built-in synchronization
- Detailed information about relevant objects

Using the Agent for Microsoft Terminal Server Features

The Agent for Microsoft Terminal Server provides enhancements to the normal RDP functionality. The following sections describe these enhancements and provide details and script samples:

Object Detail Recording

When the Agent for Microsoft Terminal Server is installed, VuGen can record specific information about the object that is being used instead of general information about the action. For example, VuGen generates **Sync Object Mouse Click** and **Sync Object Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows a double-mouse-click action recorded with and without the agent installation. Note that with the agent, VuGen generates sync_object functions for all of the mouse actions.

```
rdp_sync_object_mouse_double_click("StepDescription=Mouse Double Click on
Synchronized Object 1",
        "Snapshot=snapshot_12.inf",
        "WindowTitle=RDP2",
        "Attribute=TEXT",
        "Value=button1",
        "MouseX=100",
        "MouseY=71",
        "MouseButton=LEFT_BUTTON",
        RDP_LAST);

rdp_mouse_double_click("StepDescription=Mouse Double Click 1",
        "Snapshot=snapshot_2.inf",
        "MouseX=268",
        "MouseY=592",
        "MouseButton=LEFT_BUTTON",
        "Origin=Default",
        RDP_LAST);
```

Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When the agent is not active, you are limited to only inserting **Mouse Click**, **Mouse Double Click**, and **Sync on Image** steps.

When the agent is installed, you are able to insert all possible steps that involve the RDP agent. Below you can find explanations about some of them:

- **Get Object Info** and **Sync on Object Info**. Provide information about the state of the object and synchronize on a specific object property such as: ENABLED, FOCUSED, CONTROL_ID, ITEM_TEXT, TEXT, CHECKED, and LINES.

In the following example, the rdp_sync_on_object_info function provides synchronization by waiting for the Internet Options dialog box to come into focus.

```
rdp_sync_on_object_info("StepDescription=Sync on Object Info 0",
        "Snapshot=snapshot_30.inf",
        "WindowTitle=Internet Options",
        "ObjectX=172",
        "ObjectY=155",
        "Attribute=FOCUSED",
        "Value={valueParam}",
        "Timeout=10",
        "FailStepIfNotFound=No",
        RDP_LAST);
```

Tips for Using the Agent for Microsoft Terminal Server

The following section contains a list of tips and best practises for recording RDP scripts while using the Agent for Microsoft Terminal Server.

- When opening applications menus (e.g. File, Edit…) with the mouse, sync steps will sometimes fail. To avoid this issue, we recommend selecting menu items by using the keyboard when recording.

- When you add a **sync_on_object_mouse_click** step manually, the coordinates given are absolute coordinates (relating to the entire screen). To create the sync point, you need to calculate the offset in the window (relative coordinates) of the desired click location and modify the absolute coordinates accordingly for the synchronization to successfully replay.
- If a synchronization object exists at the correct location and time during replay, but is covered by another window (such as a pop-up), then the synchronization step will pass and a click will be executed on the window which is covering the synchronization point and therefore harm the script flow.
- During recording, if you want to return the AUT window to the foreground either click on the title bar or use the keyboard (ALT+TAB). If you click inside the AUT window to return it to the foreground, the RDP session may terminate unexpectedly.

Installing the Microsoft Terminal Server Agent

The installation file for the Agent for Microsoft Terminal Server is located on the product installation disk, under the **Additional Components\Agent for Microsoft Terminal Server** directory.

Note that the agent should only be installed on your RDP server machine, not Load Generator machines.

If you are upgrading the agent, make sure to uninstall the previous version before installing the next one (see uninstallation instructions below).

To install the Agent for Microsoft Terminal Server:

1. If your server requires administrator permissions to install software, log in as an administrator to the server.
2. If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

3. Locate the installation file, **Setup.exe**, on the LoadRunner DVD in the **Additional Components\Agent for Microsoft Terminal Server** directory.
4. Follow the installation wizard to completion.

Note: To use the agent, you must set the recording options before recording a Vuser script. In the Start Recording dialog box, click Options. In the Advanced Code Generation node, check **Use RDP Agent**.

To uninstall the Agent for Microsoft Terminal Server:

1. If your server requires administrator privileges to remove software, log in as an administrator to the server.
2. Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Microsoft Terminal Server** and click **Change/Remove**.

Effects and Memory Requirements

When you run RDP Vusers with the agent installed, each Vuser runs its own process of **lrrdpagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine.

33

Database Protocols

You use VuGen to record communication between a database client application and a server. The resulting script is called a Database Vuser script.

This chapter includes:

About Developing Database Vuser Scripts

When you record a database application communicating with a server, VuGen generates a Database Vuser script. VuGen supports the following database types: CtLib, DbLib, Informix, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity. Each LRD function has an **lrd** prefix and represents one or more database functions. For example, the **lrd_fetch** function represents a fetch operation.

When you run a recorded session, the Vuser script communicates directly with the database server, performing the same operations as the original user. You can set the Vuser behavior (run-time settings) to indicate the number of times to repeat the operation and the interval between the repetitions. For more information, see Chapter 79, "Configuring Run-Time Settings."

Using VuGen, you can parameterize a script, replacing recorded constants with parameters. For more information, see Chapter 70, "Working with VuGen Parameters."

In addition, you can correlate queries or other database statements in a script, linking the results of one query with another. For more information, see Chapter 8, "Correlating Statements."

For troubleshooting information and scripting tips, see the debugging tips.

Introducing Database Vusers

Suppose that you have a database of customer information that is accessed by customer service personnel located throughout the country. You use Database Vusers to emulate the situation in which the database server services many requests for information. A Database Vuser could:

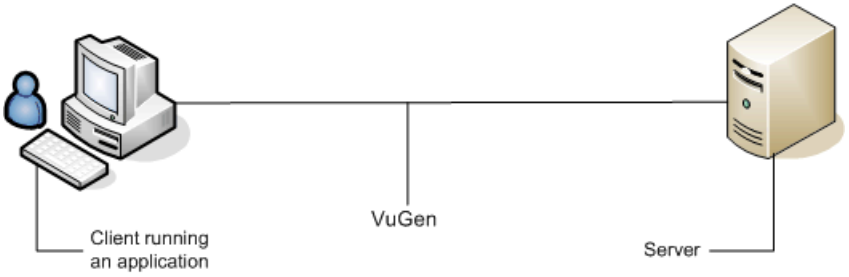
- connect to the server
- submit an SQL query
- retrieve and process the information
- disconnect from the server

You distribute several hundred Database Vusers among the available load generators, each Vuser accessing the database by using the server API. This enables you to measure the performance of your server under the load of many users.

The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. The Vusers execute the script and emulate user load on the client/server system. The Vusers generate performance data which you can analyze in report and graph format.

Understanding Database Vuser Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.



Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.

You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and UNIX environments. For information about recording scripts, see Chapter 5, "Recording with VuGen."

Users working in a UNIX only environment can create Database Vuser scripts through programming using VuGen templates as the basis for a script. For information about programming Database Vuser scripts on UNIX, see Appendix 88, "Programming Scripts on UNIX Platforms."

Getting Started with Database Vuser Scripts

This section provides an overview of the process of developing Database Vuser scripts using VuGen.

To develop a Database Vuser script:

1 Record the basic script using VuGen.

Invoke VuGen and create a new Vuser script. Specify the type of Vuser (**Client Server** or **ERP** protocol types). Select an application to record and set the recording options. Record typical operations on your application.

For details, see Chapter 5, "Recording with VuGen."

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same query many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

4 Correlate queries (optional).

Correlating database statements allows you to use the result of a query in a subsequent one. This feature is useful when working on a database with user constraints.

For details, see Chapter 8, "Correlating Statements."

5 Configure the run-time settings.

The run-time settings control the Vuser script behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using LRD Functions

The functions developed to emulate communication between a database client and a server are called LRD Vuser functions. Each LRD Vuser function has an **lrd** prefix. VuGen automatically records most of the LRD functions listed in this section during a database session (CtLib, DbLib, Informix, Oracle (2-Tier), and ODBC). You can also manually program any of the functions into your script.

The LRD functions are classified into several categories: Access Management Functions, Environment Functions, Retrieval Handling, Statement Handling, Statement Correlating, Variable Handling, Siebel, and Oracle 8.

For syntax and examples of the LRD functions, see the *Online Function Reference* (**Help > Function Reference**).

Understanding Database Vuser Scripts

After you record a database session, you can view the recorded code in VuGen's built-in editor. You can scroll through the script, see the SQL statements that were generated by your application, and examine the data returned by the server.

The VuGen window provides you with the following information about the recorded database session:

- the sequence of functions recorded
- grids displaying the data returned by database queries
- the number of rows fetched during a query

Function Sequence

When you view a Vuser script in the VuGen window, you see the sequence in which VuGen recorded your activities. For example, the following sequence of functions is recorded during a typical Oracle database session:

lrd_init	Initializes the environment.
lrd_open_connection	Connects to the database server.
lrd_open_cursor	Opens a database cursor.
lrd_stmt	Associates an SQL statement with a cursor.
lrd_bind_col	Binds a host variable to a column.
lrd_exec	Executes an SQL statement.
lrd_fetch	Fetches the next record in the result set.
lr_commit	Commits a database transaction.
lr_close_cursor	Closes a cursor.
lrd_close_connection	Disconnects from the database server.
lrd_end	Cleans up the environment.

In the following script, VuGen recorded the actions of an operator who opened a connection to an Oracle server and then performed a query requesting the local settings.

```
lrd_init(&InitInfo, DBTypeVersion);
lrd_open_connection(&Con1, LRD_DBTYPE_ORACLE, "s1", "tiger", "hp1", "", 0, 0, 0);
lrd_open_cursor(&Csr1, Con1, 0);
lrd_stmt(Csr1, "select parameter, value     from v$nls_parameters "
     "    where (upper(parameter) in ('NLS_SORT','NLS_CURRENCY',"
     "'NLS_ISO_CURRENCY', 'NLS_DATE_LANGUAGE',"
     "'NLS_TERRITORY'))", -1, 0 /*Non deferred*/, 1 /*Dflt Ora Ver*/, 0);
lrd_bind_col(Csr1, 1, &D1, 0, 0);
lrd_bind_col(Csr1, 2, &D2, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_fetch(Csr1, 7, 7, 0, PrintRow2, 0);
…
lrd_close_cursor(&Csr1, 0);
lrd_commit(0, Con1, 0);
lrd_close_connection(&Con1, 0, 0);
lrd_end(0);
```

Row Information

VuGen generates an lrd_fetch function for each SQL query.

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

The second parameter of the function indicates the number of rows fetched. This number can be positive or negative.

Positive Row Values

A positive value shows the number of rows fetched during recording, and indicates that not all rows were fetched. (For example, if the operator cancelled the query before it was completed.)

In the following example, four rows were retrieved during the database query, but not all of the data was fetched.

```
lrd_fetch(Csr1, 4, 1, 0, PrintRow7, 0);
```

During execution, the script retrieves the number of rows indicated by the positive value (provided the rows exist).

Negative Row Values

A negative row value indicates that all available rows were fetched during recording. The absolute value of the negative number is the number of rows fetched.

In the following example, all four rows of the result set were retrieved:

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

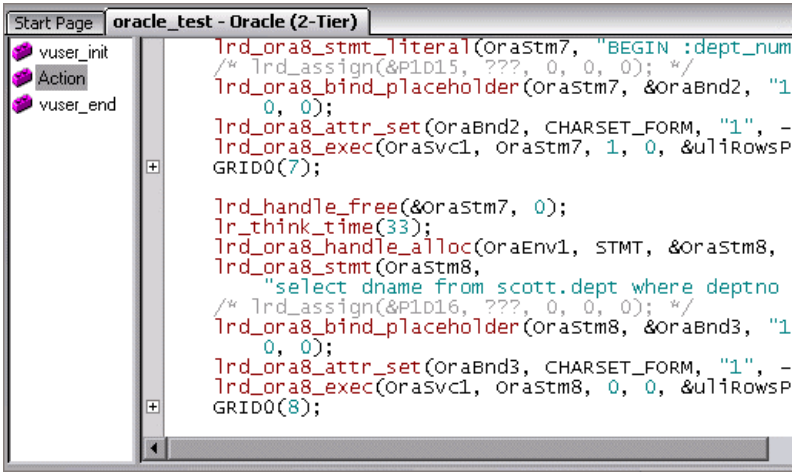
When you execute an **lrd_fetch** statement containing a negative row value, it retrieves all of the available rows in the table at the time of the run—not necessarily the number at the time of recording. In the above example, all four rows of the table were retrieved during the recording session. However, if more rows are available during script execution, they are all retrieved.

For more information about **lrd_fetch**, see the *Online Function Reference* (**Help** > **Function Reference**).

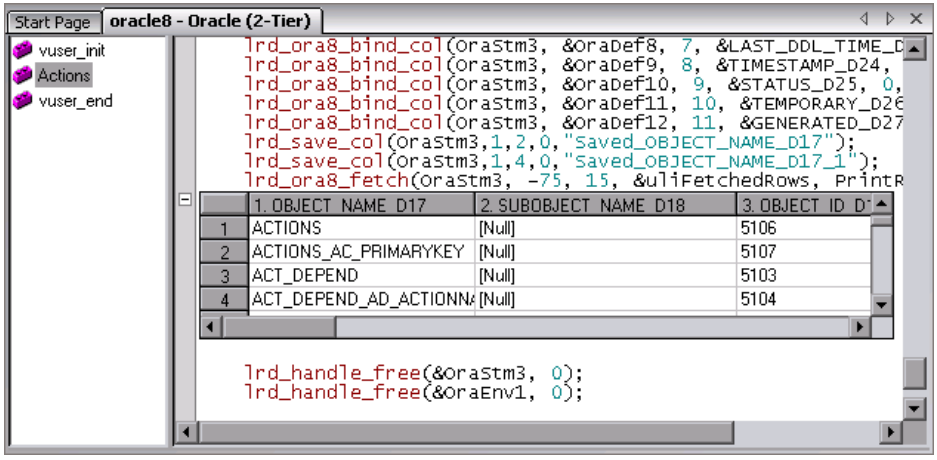
Working with Grids

The data returned by a query during a recording session is displayed in a grid. By viewing the grid you can determine how your application generates SQL statements and the efficiency of your client/server system.

The data grid is represented by a **GRID** statement. To open the data grid, click on the icon in the margin adjacent to the GRID statement.



In the following example, VuGen displays a grid for a query executed on a flights database. The query retrieves the flight number, airport code, departure city, day of the week, and other flight-relevant information.



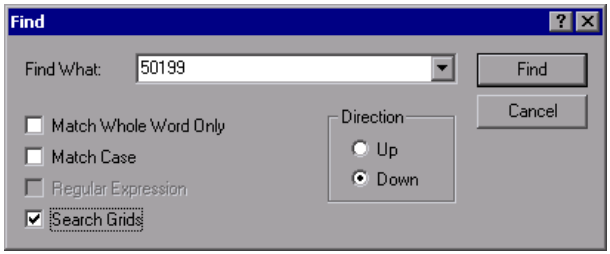
If the data value is very long, only part of it is shown in the grid. This truncation only occurs in the displayed grid and has no impact on the data.

The grid columns are adjustable in width. You can scroll up to 200 rows using the scroll bar. To change this value, open the **vugen.ini** file on your machine's Operating System folder (for example, C:\WINNT) and modify the following entry:

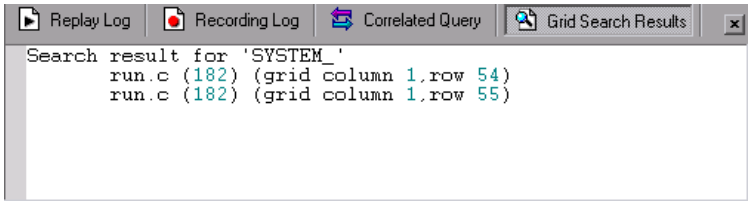
```
[general]
max_line_at_grid=200
```

To correlate a value or save the data to a file, click in a cell and use the right-click menu options, **Create Correlation** or **Save To File**.

To search for data within the entire grid, including the non-visible part, Select **Search Grids** in the Find dialog box.



VuGen displays the results in the Output window's **Grid Results** tab.



Troubleshooting Database Protocols

The following section contains troubleshooting information for database protocols.

IE crashes when recording Oracle NCA/11i scripts

This can occur due to an incompatible dll file.

To replace the incompatible dll:

1. Open the **C:\program files\oracle\JInitiator_1.3.1.18\bin\hotspot** directory.
2. Back up the **jvm.dll** file.
3. Delete the **jvm.dll** file and replace it with a different version of the file.

Evaluating Error Codes

When a Vuser executes an LRD function, the function generates a return code. A return code of 0 indicates that the function succeeded. For example, a return code of 0 indicates that another row is available from the result set. If an error occurs, the return code indicates the type of error. For example, a return code of 2014 indicates that an error occurred in the initialization.

There are four types of return codes, each represented by a numerical range:

Type of Return Code	Range
Informational	0 to 999
Warning	1000 to 1999
Error	2000 to 2999
Internal Error	5000 to 5999

For more detailed information on the return codes, see the *Online Function Reference* (**Help > Function Reference**).

You can evaluate the return code of an LRD function to determine if the function succeeded. The following script segment evaluates the return code of an lrd_fetch function:

```
static int rc;
rc=lrd_fetch(Csr15, -13, 0, 0, PrintRow4, 0);
if (rc==0)
    lr_output_message("The function succeeded");
else
    lr_output_message("The function returned an error code:%d",rc);
```

Handling Errors

You can control how database Vusers handle errors when you run a database Vuser script. By default, if an error occurs during script execution, the script execution is terminated. To change the default behavior, you can instruct the Vuser to continue when an error occurs. You can apply this behavior in the following ways:

- Globally Modifying Error Handling. Provides error handling to the entire script, or to a segment of the script
- Locally Modifying Error Handling. Provides error handling to a specific function only

Globally Modifying Error Handling

You can change the way that Vusers handle errors by issuing an LRD_ON_ERROR_CONTINUE or LRD_ON_ERROR_EXIT statement. By default, a Vuser aborts the script execution when it encounters any type of error—database, parameter related, and so on. To change the default behavior, insert the following line into your script:

```
LRD_ON_ERROR_CONTINUE;
```

From this point on, the Vuser continues script execution, even when an error occurs.

You can also specify that the Vuser continue script execution when an error occurs only within a segment of the script. For example, the following code tells the Vuser to continue script execution even if an error occurs in the lrd_stmt or lrd_exec functions:

```
LRD_ON_ERROR_CONTINUE;
lrd_stmt(Csr1, "select…"…);
lrd_exec(…);
LRD_ON_ERROR_EXIT;
```

Use the LRD_ON_ERROR_CONTINUE statement with caution, as significant and severe errors may be missed.

Locally Modifying Error Handling

You can set error handling for a specific function by modifying the severity level. Functions such as **lrd_stmt** and **lrd_exec**, which perform database operations, use severity levels. The severity level is indicated by the function's final parameter, **miDBErrorSeverity**. This parameter tells the Vuser whether or not to continue script execution when a database error occurs (error code 2009). The default, 0, indicates that the Vuser should abort the script when an error occurs.

For example, if you reference a table that does not exist, the following database statement fails and the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
        1 /*Dflt Ora Ver*/, 0);
```

To tell a Vuser to continue script execution, even when a database operation error occurs for that function, change the statement's severity parameter from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
        1 /*Dflt Ora Ver*/, 1);
```

When the severity is set to 1 and a database error occurs, a warning is issued. Note that the severity level set for a particular function applies only to that function.

CtLib Result Set Errors

In CtLib recording, the application retrieves all of the available result sets after executing a statement. If the returned result set contains fetchable data, the application performs bind and fetch operations on the data as indicated in the following example:

```
lrd_stmt(Csr15, "select * from all_types", -1, 148, -99999, 0);
lrd_exec(Csr15, 0, 0, 0, 0, 0);
lrd_result_set(Csr15, 1 /*Succeed*/, 4040 /*Row*/, 0);
lrd_bind_col(Csr15, 1, &tinyint_D41, 0, 0);
…
lrd_fetch(Csr15, -9, 0, 0, PrintRow3, 0);
```

If a result set does not contain fetchable data, bind and fetch operations cannot be performed.

When you parameterize your script, result data may become unfetchable (depending on the parameters). Therefore, a CtLib session that recorded bind and fetch operations for a particular statement, may not be able to run, if the new data is unfetchable. If you try to execute an **lrd_bind_col** or an **lrd_fetch** operation, an error will occur (LRDRET_E_NO_FETCHABLE_DATA — error code 2064) and the Vuser will terminate the script execution.

You can override the error by telling the Vuser to continue script execution when this type of error occurs. Insert the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_CONT;
```

To return to the default mode of terminating the script execution, type the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_EXIT;
```

Use this option with caution, as significant and severe errors may be missed.

34

Database - Script Correlation

After you record a database session, you may need to correlate one or more queries within your script—use a value that was retrieved during the database session, at a later point in the session.

This chapter includes:

About Correlating Database Vuser Scripts

If you encounter an error when running your script, examine the script at the point where the error occurred. In many cases, you can overcome the problem by correlating the query. Correlating the query means that you save a run-time value to a parameter. You then use the saved value at a later point in the same script. In summary, correlation is using the results of one statement as input to another.

There are many queries whose inputs depend on the result of prior queries. To emulate this behavior, use VuGen's correlation capabilities.

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script and allow a successful replay. It performs the following steps:

- Scans for potential correlations
- Inserts the appropriate correlation function to save the results to a parameter
- Replaces the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

To scan and correlate a script detected with automatic correlation:

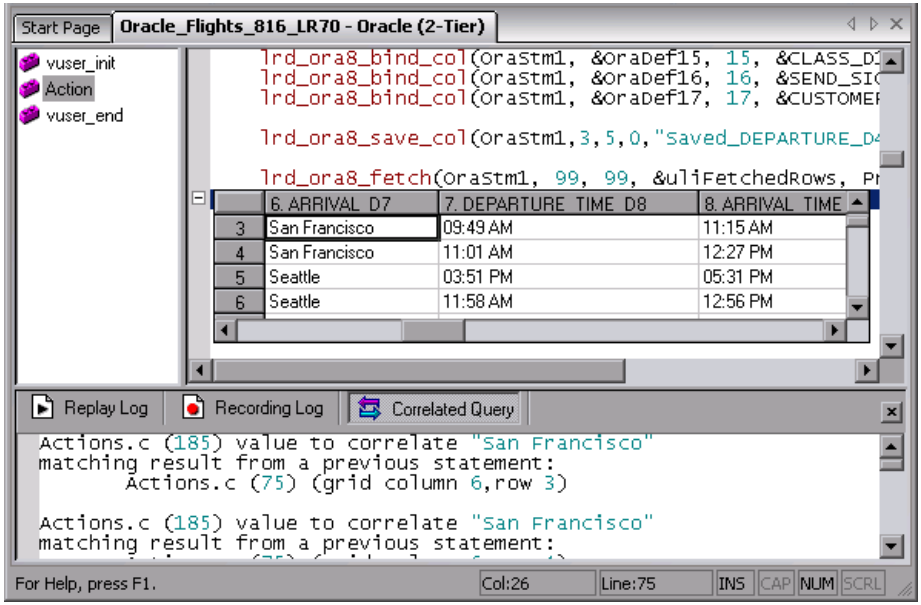
1 Open the Output window.

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

2 Select **Vuser** > **Scan for Correlations**.

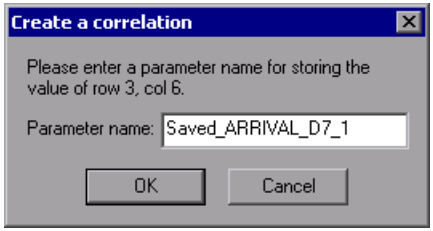
VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.

In the following example, in the **lrd_ora8_fetch** function, VuGen detected a value to correlate.



3 In the Correlated Query tab, double-click on the result you want to correlate. Click on the words **(grid column x, row y)** VuGen sends the cursor to the location of the value in your grid.

4 Select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



5 Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**, **lrd_ora8_save_col**) which saves the result to a parameter.

6 Click **Yes** to confirm the correlation.

A message box opens asking if you want to search for all occurrences of the value in the script.

➤ To replace only the value in the selected statement, click **No**.

➤ To search and replace additional occurrences, click **Yes**.

7 A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

8 Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. If you cancel the correlation, VuGen also erases the statement created in the previous step.

Correlating a Known Value

If you know which value needs to be correlated, perform the following procedure.

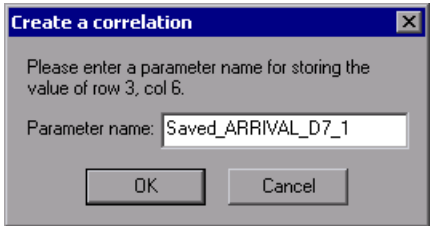
To correlate a specific value:

1 Locate the statement in your script, with the query containing the value you want to correlate. This is usually one of the arguments of the **lrd_assign**, **lrd_assign_bind**, or **lrd_stmt** functions. Select the value without the quotation marks.

2 Select **Scan for Correlations (at cursor)** from the right-click menu. VuGen scans the selected value for correlations.

3 In the Output window's **Correlated Query** tab, double-click on the result you want to correlate. Click on the words (**grid column x, row y**). VuGen sends the cursor to the location of the value in your grid.

4 In the grid, click on the value you want to correlate and select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



5 Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**, **lrd_ora8_save_col**) which saves the result to a parameter.

6 Click **Yes** to confirm the correlation.

A message box opens asking if you want to search for all occurrences of the value in the script.

- ➤ To replace only the value in the selected statement, click **No**.
- ➤ To search and replace additional occurrences, click **Yes**.

7 A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

8 Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. If you cancel the correlation, VuGen also erases the statement created in the previous step.

Note: If you are correlating a value from an **lrd_stmt** function, the following data types are not supported: date, time, and binary (RAW, VARRAW).

Database Correlation Functions

When working with Database Vuser scripts, (DbLib, CtLib, Oracle, Informix, and so forth) you can use VuGen's automated correlation feature to insert the appropriate functions into your script. The correlating functions are:

- **lrd_save_col** saves a query result appearing in a grid, to a parameter. This function is placed before fetching the data. It assigns the value retrieved by the subsequent **lrd_fetch** to the specified parameter. (**lrd_ora8_save_col** for Oracle 8 and higher)
- **lrd_save_value** saves the current value of a placeholder descriptor to a parameter. It is used with database functions that set output placeholders (such as certain stored procedures under Oracle).
- **lrd_save_ret_param** saves a stored procedure's return value to a parameter. It is used primarily with database procedures stored in DbLib that generate return values.

Note: VuGen does not apply correlation if the saved value is invalid or NULL (no rows returned).

For more information about these functions and their arguments, see the the *Online Function Reference* (**Help > Function Reference**).

35

DNS Protocol

VuGen allows you to emulate network activity by directly accessing a DNS server.

This chapter includes:

About Developing DNS Vuser Scripts

The DNS protocol is a low-level protocol that allows you to emulate the actions of a user working against a DNS server.

The DNS protocol emulates a user accessing a Domain Name Server to resolve a host name with its IP address. Only replay is supported for this protocol—you need to manually add the functions to your script.

To create a script for the DNS protocol, select **File** > **New** to open the New Virtual User dialog box. Select the Domain Name Resolution (DNS) protocol type from the Client/Server category. Since recording is not supported for DNS, you program the script with the appropriate DNS, Vuser API and C functions. For more information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

After you create a Vuser script, you integrate it into a scenario on either a Windows or UNIX platform. For more information on integrating Vuser scripts in a scenario, see the *HP LoadRunner Controller User's Guide*.

Working with DNS Functions

DNS Vuser script functions record queries to and from a Domain Name Resolution (DNS) server. Each DNS function begins with a **dns** prefix. For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Function Name	Description
ms_dns_query	Resolves the IP address of a host.
ms_dns_nextresult	Advances to the next IP address in the list returned by **ms_dns_query**.

In the following example, a query is submitted to the DNS server and the results are printed to the log file.

```
Actions()
{
int    rescnt = 0;
     char results = NULL;
     results = (char *) ms_dns_query("transaction",
                                     "URL=dns://<DnsServer>",
                                     "QueryHost=<Hostname>",
                                     LAST);

     // List all the IP addresses of the host names...
     while (*results) {
          rescnt++;
          lr_log_message(lr_eval_string("(%d) IP of<Hostname> is %s"),
               rescnt, results);
          results = (char *) ms_dns_nextresult(results);
     }
     return 1;
}
```

36

Windows Sockets (WinSock) Protocol

You use VuGen to record communication between a client application and a server that communicate using the Windows Sockets protocol. The resulting script is called a Windows Sockets Vuser script.

This chapter includes:

About Recording Windows Sockets Vuser Scripts

The Windows Sockets protocol is ideal for analyzing the low level code of an application. For example, to check your network, you can use a Windows Sockets (WinSock) script to see the actual data sent and received by the buffers. The WinSock type can also be used for recording other low level communication sessions. In addition, you can record and replay applications that are not supported by any of the other Vuser types.

When you record an application which uses the Windows Sockets protocol, VuGen generates functions that describe the recorded actions. Each function begins with an **lrs** prefix. The LRS functions relate to the sockets, data buffers, and the Windows Sockets environment. Using VuGen, you record your application's API calls to the Winsock.dll or Wsock32.dll.

For example, you could create a script by recording the actions of a *telnet* application.

In the following example, lrs_send sends data to a specified socket:

```
lrs_send("socket22", "buf44", LrsLastArg);
```

You can view and edit the recorded script from VuGen's main window. The Windows Sockets API calls that were recorded during the session are displayed in the window, allowing you to track your network activities.

VuGen can display a WinSock script in two ways:

- As an icon-based representation of the script. This is the default view, and is known as the **Tree view**.
- As a text-based representation of the script showing the Windows Sockets API calls. This is known as the **Script view**.

You use VuGen to view and edit your Vuser script from either the Tree view or Script view. For more information, see "Viewing and Modifying Vuser Scripts" on page 48.

After creating a script, you can view the recorded data as a snapshot or as a raw data file. For details, see "Working with Windows Socket Data" on page 608

Getting Started with Windows Sockets Vuser Scripts

This section provides an overview of the process of developing Windows Sockets Vuser scripts using VuGen.

To develop a Windows Sockets script:

1 Record the actions using VuGen.

Invoke VuGen and create a new Vuser script, specifying Windows Sockets as the type. Select an application to record and set the recording options. Record typical operations on your application.

For details, see Chapter 5, "Recording with VuGen."

2 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, "Correlating Statements."

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings." and Chapter 80, "Configuring Network Run-Time Settings."

6 Run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

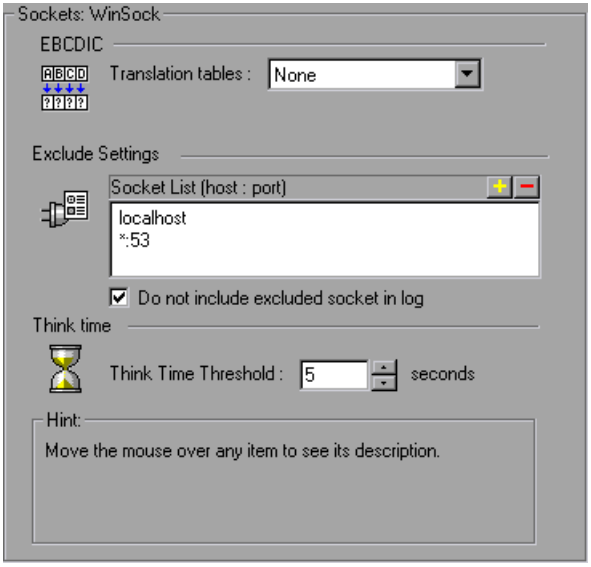
After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Setting the WinSock Recording Options

The following recording options are available for WinSock Vusers:

- Configuring the Translation Table
- Excluding Sockets
- Setting the Think Time Threshold

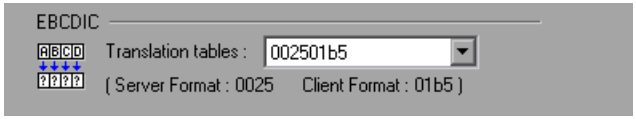
To open the Recording Options dialog box, select **Tools** > **Recording Options** or click the **Options** button in the Start Recording dialog box. VuGen displays the WinSock options.



Configuring the Translation Table

To display data in EBCDIC format, you specify a translation table in the recording options.

The Translation Table lets you specify the format for recording sessions. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. Select a translation option from the list box.



The first four digits of the listbox item represent the server format. The last four digits represent the client format. In the above example, the selected translation table is 002501b5. The server format is 0025 and the client format is 01b5 indicating a transfer from the server to the client. In a transmission from the client to the server, you would select the item that reverses the formats—01b50025 indicating that the client's 01b5 format needs to be translated to the server's 0025 format.

The translation tables are located in the **ebcdic** directory under the VuGen's installation directory. If your system uses different translation tables, copy them to the **ebcdic** directory.

Note: If your data is in ASCII format, it does not require translation. You must select the **None** option, the default value. If you do select a translation table, VuGen will translate the ASCII data.

When working on Solaris machines, you must set the following environment variables on all machines running the Vuser scripts:

```
setenv LRSDRV_SERVER_FORMAT 0025
setenv LRSDRV_CLIENT_FORMAT 04e4
```

Excluding Sockets

VuGen supports the Exclude Socket feature, allowing you to exclude a specific socket from your recording session. To exclude all actions on a socket from your script, you specify the socket address in the Exclude Socket list. To add a socket to the list, click the plus sign in the upper right corner of the box and enter the socket address in one of the following formats:

Value	Meaning
host:port	Exclude only the specified port on the specified host.
host	Exclude all ports for the specified host.
:port	Exclude the specified port number on the local host.
*:port	Exclude the specified port number on all hosts.

You can exclude multiple hosts and ports by adding them to the list. To remove a socket from the excluded list, select the socket address and click the minus sign in the upper right corner of the box. We recommend that you exclude hosts and ports that do not influence the server load under test, such as the local host and the DNS port (53), which are excluded by default.

By default, VuGen does not log the actions of the excluded sockets in the Excluded Socket List. To instruct VuGen to log the actions of the excluded socket(s) clear the Do not include excluded sockets in log check box. When logging is enabled for the excluded sockets, their actions are preceded by "Exclude" in the log file.

```
Exclude : /* recv(): 15 bytes were received from socket 116 using flags 0 */
```

Setting the Think Time Threshold

During recording, VuGen automatically inserts the operator's think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an **lr_think_time** statement before LRS functions. If the recorded think time is below the threshold level, an **lr_think_time** statement is not generated.

To set the think time threshold, enter the desired value (in seconds) in the **Think Time Threshold** box. The default value is five seconds.

Using LRS Functions

The functions developed to emulate communication between a client and a server by using the Windows Sockets protocol are called LRS Vuser functions. Each LRS Vuser function has an **lrs** prefix. VuGen automatically records most of the LRS functions listed in this section during a Windows Sockets session. You can also manually program any of the functions into your Vuser script.

The LRS functions are classified into several categories: Socket, Buffer, Environment, Correlating Statements, Conversion, and Timeout.

For more information about the LRS functions, see the *Online Function Reference* (**Help > Function Reference**).

After you record a session, VuGen's built-in editor lets you view the recorded code. You can scroll through the script, view the functions that were generated by your application, and examine the transferred data. When you view the script in the main window, you see the sequence in which VuGen recorded your activities.

The following function sequence is recorded during a typical session:

lrs_startup	Initializes the WinSock DLL.
lrs_create_socket	Initializes a socket.
lrs_send	Sends data on a datagram or to a stream socket.
lrs_receive	Receives data from a datagram or stream socket.
lrs_disable_socket	Disables an operation on a socket.
lrs_close_socket	Closes an open socket.
lrs_cleanup	Terminates the use of the WinSock DLL.

VuGen supports record and replay for applications using the Windows Socket protocol on Windows; on UNIX platforms, only replay is supported.

Working with Windows Socket Data

After you record an application using VuGen, you have multiple data buffers containing the data.

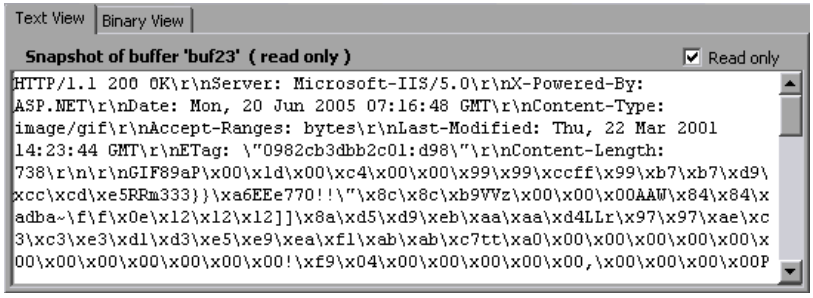
When you view the WinSocket script in tree view, VuGen provides a snapshot window which allows you to navigate within the data buffers and modify the data.

When working in script view, you can view the raw data in the **data.ws** file. For more information, see "Viewing Windows Socket Data in Script View" on page 622.

Viewing Data in the Snapshot Window

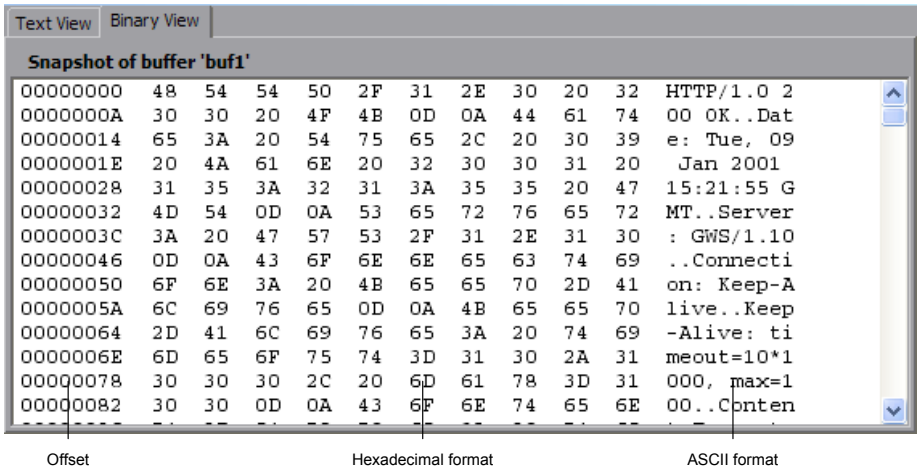
When viewing a Windows Socket script in tree view, VuGen provides a buffer snapshot window which displays the data in an editable window. You can view a snapshot in either **Text** view or **Binary** view.

The text view shows a snapshot of the buffer with the contents represented as text.



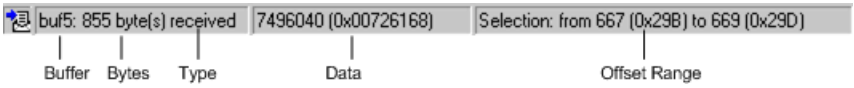
By default, VuGen stores the buffer data as read-only data. If you want to modify the contents of the buffer, clear the **Read only** box in the buffer's Text View. VuGen issues a warning that bookmarks and parameters may be affected.

The binary view shows the data in hexadecimal representation. The left column shows the offset of the first character in that row.
The middle column displays the hexadecimal values of the data. The right column shows the data in ASCII format.



The status bar below the buffer snapshot provides information about the data and buffer:

- **Buffer number.** The buffer number of the selected buffer.
- **Total bytes.** the total number of bytes in the buffer.
- **Buffer type.** the type of buffer—received or sent.
- **Data.** the value of the data at the cursor in decimal and hexadecimal formats, in Little Endian order (reverse of how it appears in the buffer).
- **Offset.** the offset of the selection (or cursor in text view) from the beginning of the buffer. If you select multiple bytes, it indicates the range of the selection.



The status bar also indicates whether or not the original data was modified.

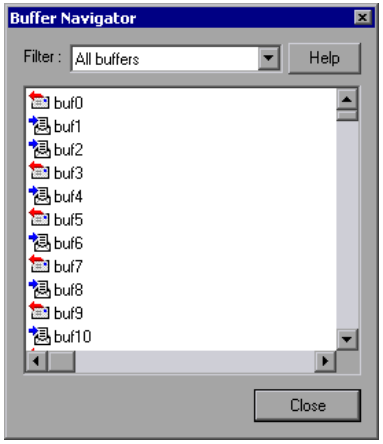
Navigating Through the Data

In tree view, VuGen provides several tools that allow you to navigate through the data in order to identify and analyze a specific value:

- Buffer Navigator
- Go To Offset
- Bookmarks

Buffer Navigator

By default, VuGen displays all the steps and buffers in the left pane. The Buffer Navigator is a floating window that lets you display only the receive and send buffers steps (**lrs_send**, **lrs_receive**, **lrs_receive_ex**, and **lrs_length_receive**). In addition, you can apply a filter and view either the send or receive buffers.



When you select a buffer in the navigator, its contents are displayed in the buffer snapshot window.

If you change a buffer's name after recording, its contents will not appear in the snapshot window when you click on the step. To view the renamed buffer's data, use the buffer navigator and select the new buffer's name. VuGen issues a warning message indicating that parameter creation will be disabled for the selected buffer.

To open the Buffer Navigator, select **View > Buffer Navigator**. To close the navigator, click the X in the top right corner of the navigator dialog box.

Note that you can also navigate between buffers by clicking on the buffer step in the left pane's tree view. The advantages of the buffer navigator are that it is a floating window with filtering capabilities.

Go To Offset

You can move around within the data buffer by specifying an offset. You can indicate the absolute location of the data, or a location relative to the current position of the cursor within the buffer. This dialog box also lets you select a range of data, by specifying the starting and end offsets.

To go to an offset:

1 Click within the snapshot window. Then select **Go to offset** from the right-click menu. The Go to offset dialog box opens.

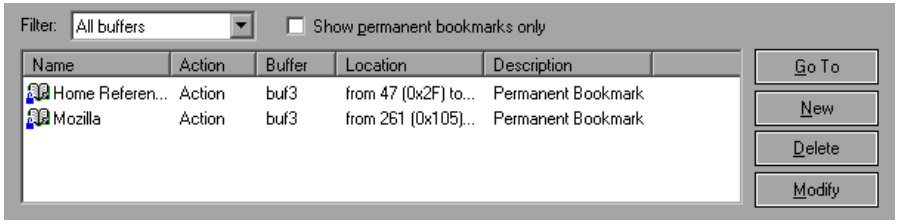
2 To go to a specific offset within the buffer (absolute), click **Go to offset** and specify an offset value.

3 To jump to a location relative to the cursor, click **Advance by** and specify the number of bytes you want to advance. To advance ahead, enter a positive value. To move backwards within the buffer, use a negative value.

4 To select a range of data within the buffer, click **Select range from** and specify the beginning and end offsets.

Bookmarks

VuGen lets you mark locations within a buffer as bookmarks. You give each bookmark a descriptive name and click on it to jump directly to its location. The bookmarks are listed in the Output window's **Bookmarks** tab below the buffer snapshot.



Bookmarks can be used in both the text and binary views. You can locate the desired data in text view, save the location as a bookmark, and jump directly to that bookmark in binary view.

The bookmark can mark a single byte or multiple bytes. When you click on a bookmark in the list, it is indicated in the buffer snapshot window as a selection. Initially, in the text view the data is highlighted in blue, and in binary view the bookmark block is marked in red. Also in binary view, when you place your cursor over a bookmark, a popup text box opens indicating the name of the bookmark.

You can create both permanent and simple bookmarks. A permanent bookmark is always marked within the buffer's binary view—it is enclosed by a blue box. The bookmark stays selected in blue, even when pointing to another location in the buffer. The cursor location is marked in red. A simple bookmark, however, is not permanently marked. When you jump to a simple bookmark, it is marked in red, but once you move the cursor within the buffer, the bookmark is no longer selected. By default bookmarks are permanent.

To work with bookmarks:

1. To create a bookmark, select one or more bytes in a buffer snapshot (text or binary view) and select **New Bookmark** from the right-click menu.
2. To view the bookmark list, select **View** > **Output Window** and select the **Bookmarks** tab.
3. To assign a name to a bookmark, click on it in the bookmark list and edit the title.
4. To change the location of a bookmark, select the bookmark in the **Bookmarks** tab, then select the new data in the buffer snapshot. Click **Modify** in the **Bookmarks** tab.
5. To change a bookmark form being Permanent to simple (permanent means that it is always marked, even when you move the cursor to a new location), select the bookmark, perform a right-click, and clear the check adjacent to **Permanent Bookmark**.
6. To display only permanent bookmarks in the list, select the **Show Permanent Bookmarks only** check box in the **Bookmarks** tab.
7. To view bookmarks from a specific buffer, select a bookmark from the desired buffer and select **Selected buffer only** in the **Filter** box.
8. To delete a bookmark, select it in the **Bookmarks** tab and click **Delete**.

Modifying Buffer Data

In tree view, VuGen provides several tools that allow you to modify the data by deleting, changing or adding to the existing data.

- Inserting Data
- Editing Data
- Parameterizing the Data

Inserting Data

You can insert a numerical value into a data buffer. You can insert it as a single, double-byte or 4-byte value.

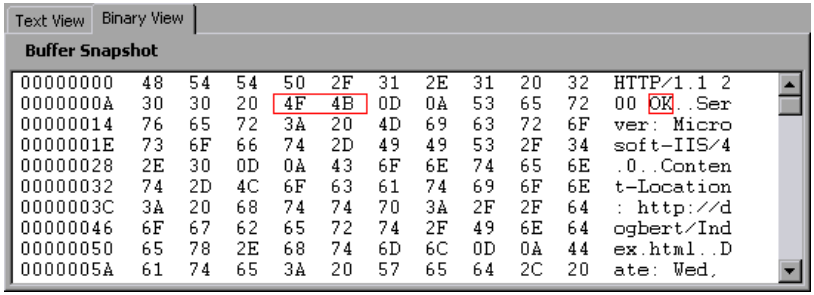
To insert a number into a data buffer:

1. Click at a location in the buffer.
2. Open the right-click menu and select **Advanced** > **Insert Number** > **Specify...**
3. Enter the ASCII value that you want to insert into the **Value** box.
4. Select the size of the data you want to insert: 1 byte, 2 bytes, or 4 bytes from the **Size** box.
5. Click **OK** to finish. VuGen inserts the hexadecimal representation of the data into the buffer.

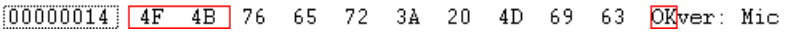
Editing Data

You can perform all of the standard edit operation on buffer data: copy, paste, cut, delete, and undo. In the binary view you can specify the actual data to insert. VuGen allows you to specify the format of the data—single byte, 2-byte, or 4-byte, and hexadecimal or decimal value. You can copy binary data and insert it as a number into the buffer. You can see the decimal or hexadecimal numbers in the right column of the binary view.

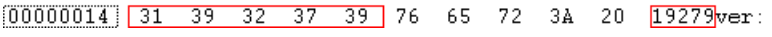
In the following example, the word **OK** was selected.



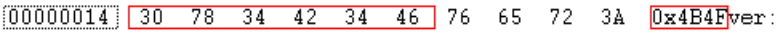
If you perform simple copy (CTRL+C) and paste (CTRL+V) operations at the beginning of the next line of data, it inserts the actual text.



If you select **Advanced Copy as Number** > **Decimal** and then paste the data, it inserts the decimal value of the ASCII code of the selected characters:



If you select **Advanced Copy as Number** > **Hexadecimal** and then paste the data, it inserts the hexadecimal value of the ASCII code of the selected characters:



The Undo Buffer retains all of the modifications to the buffer. This information is saved with the file—if you close the file it will still be available. If you want to prevent others from undoing your changes, you can empty the Undo buffer. To empty the Undo buffer, select **Advanced** > **Empty Undo Buffer** in the right-click menu.

To edit buffer data in the binary view:

1 To copy buffer data:

- As characters, select one or more bytes and press CTRL+C.
- As a decimal number, **Advanced** > **Copy As Number** > **Decimal** in the right-click menu.
- As a hexadecimal number, **Advanced** > **Copy As Number** > **Hexadecimal** in the right-click menu.

2 To paste the data:

- As a single byte (assuming the size of the data on the clipboard is a single byte), click at the desired location in the buffer and press CTRL+V.
- In short format (2-byte), **Advanced** >**Insert Number** >**Paste Short (2-byte)** in the right-click menu.
- In long format (4-byte), **Advanced** >**Insert Number** >**Paste Long (4-byte)** in the right-click menu.

3 To delete data, select it in either one of the views and select **Delete** from the right-click menu.

Parameterizing the Data

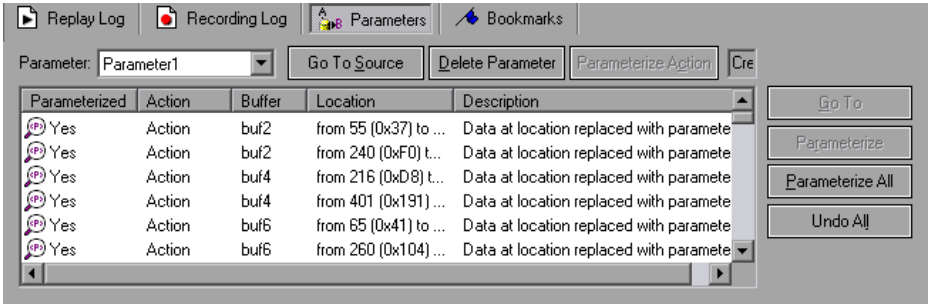
In tree view, VuGen lets you parameterize the data directly from the buffer snapshot view. You can specify a range of what to parameterize and you can specify borders. If you do not specify borders for the parameterized string, then VuGen inserts an **lrs_save_param** function into your script. If you specify borders, VuGen inserts **lrs_save_searched_string** into your script since this function allows you to specify boundary arguments.

Note that the **lrs_save_param** and **lrs_save_searched_string** functions correlate the data. This means that it stores the data that is received, for use in a later point within the test. Since correlation stores the received data, it only applies to Receive buffers and not to Send buffers. The recommended procedure is to select a string of dynamic data within the Receive buffer that you want to parameterize. Use that same parameter in a subsequent Send buffer.

This type of correlation should not be confused with simple parameterization. Simple parameterization (**Insert > New Parameter**) only applies to data within Send buffers. You set up a parameter and assign it several values. VuGen uses the different values in each of the test runs or iterations. For more information, see Chapter 70, "Working with VuGen Parameters."

The next sections discuss the correlation of data in Receive buffers.

After you create a parameter, VuGen lists all the locations in which it replaced the string with a parameter. VuGen also provides information about the creation of the parameter—the buffer in which it was created and the offset within the buffer. It lists all occurrences of the parameter in the Output Window's **Parameters** tab, below the snapshot view.

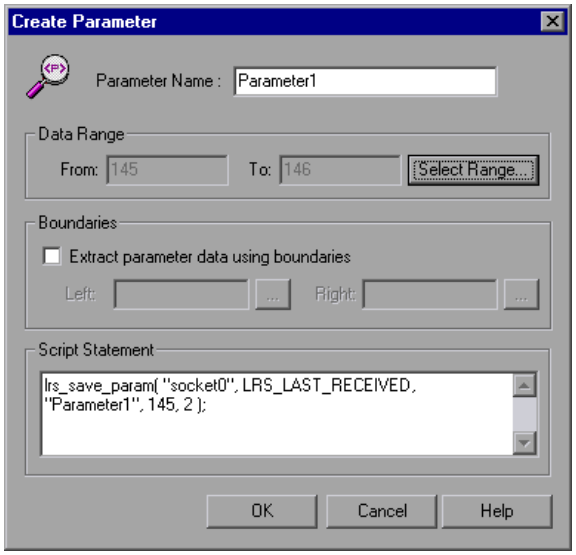


VuGen allows you to manipulate the parameters:

- **Filtering.** You can filter the parameter replacements by the parameter name.
- **Go to Source.** Select a replacement and click Go To Source to jump to the exact location of the replaced parameter within the buffer.
- **Deleting.** You can delete any one of the parameters. When you delete a parameter, VuGen replaces the data with its original value and removes the parameterization function from the script.
- **Name.** You can provide a name to each replacement.
- **Undo Replacement.** You can also undo one or more replacements displayed in the list.

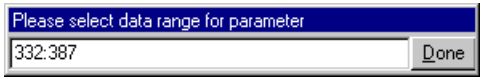
To parameterize data from the snapshot window:

1 Select the data you want to parameterize and select **Create Parameter** from the right-click menu (only available for Receive buffers). A dialog box opens:



2 Specify a name for the parameter in the **Parameter Name** box.

3 Select a range of characters to parameterize. By default, VuGen takes the range of data that you selected in the buffer. To select a range other than the one that appears in the dialog box, click **Select Range**. A small dialog box opens indicating the selected range.



Select a range in the buffer snapshot window and click **Done**.

4 If the parameter data is not constant but its borders are consistent, you can specify a right and left boundary.

To specify boundaries:

- Select the **Extract Parameter Data Using Boundaries** check box. VuGen changes the function in the **Script Statement** section from **lrs_save_param** to **lrs_save_searched_string**. Click **Done.**
- Click the browse button adjacent to the **Left** box in the **Boundaries** section. A small dialog box opens, indicating your selection within the buffer. Select the boundaries within the buffer and click **Done**. Repeat this step for the right boundary.

5. Make the desired modifications to the arguments in the **Script Statement** section. For example you can add **_ex** to the **lrs_save_param** function to specify an encoding type. For more information about these functions see the *Online Function Reference* (**Help** > **Function Reference**).
6. Click **OK** to create the parameter. VuGen asks you for a confirmation before replacing the parameter. Click **Yes**. You can view all the replacements in the **Parameters** tab.
7. To jump to the original location of the parameter within its buffer, select it and click **Go To Source**.
8. To jump to the buffer location of the selected replacement, select it and click **Go To**.
9. To delete an entire parameter, select the parameter in the **Filter** box and click **Delete Parameter**.
10. To undo a replacement, select it in the **Parameters** tab and click **Undo**. To undo all replacements of the displayed parameter, select it in the **Parameters** tab and click **Undo All.**
11. When you undo specific replacements, the Parameterized column shows **No** for that occurrence. To reapply the parameterization rule to an occurrence that was undone, select it and select **Replace With Parameter** from the right-click menu.
12. To delete an entire parameter and undo all the replacements, select the parameter in the **Filter** box and click **Delete Parameters**.
13. Select **Vuser** > **Parameter List** to assign data to the parameters.

Modifying Buffer Names

You can modify the name of a buffer using the Script view of the **data.ws** file. If you modify a buffer name after recording, this will affect the replay of the Vuser script. You can view the contents of the renamed buffer in the Script view or in Tree view using the Buffer Navigator.

If you created bookmarks in the buffer and it is not longer available, VuGen prompts you to delete the bookmarks within the buffer in which they were defined.

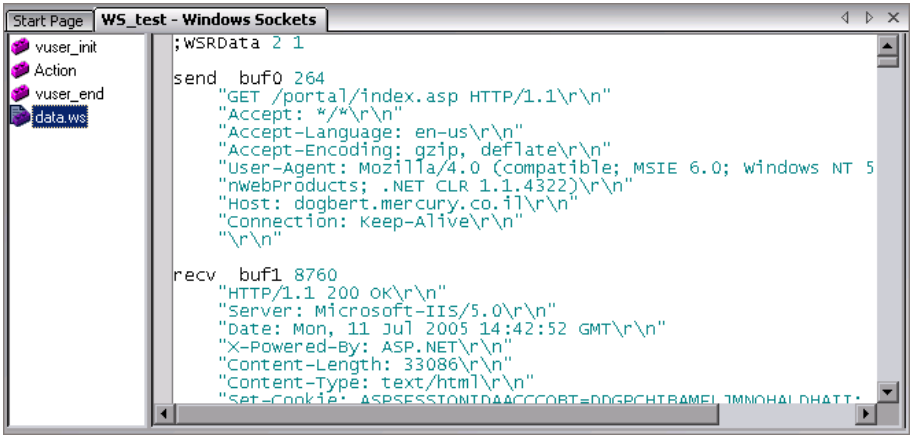
If you created parameters in the buffer and it is not longer available, VuGen prompts you to delete the parameters from the buffer in which they were defined. When you delete the parameter, all replacements are undone, even those in other buffers.

When you view the renamed buffer in the Buffer Navigator, VuGen warns you that parameter creation will be disabled within that buffer.

Viewing Windows Socket Data in Script View

When you use VuGen to create a Windows Sockets Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**. In addition to the Vuser script, VuGen also creates a data file, **data.ws** that contains the data that was transmitted or received during the recording session. You can use VuGen to view the contents of the data file by selecting **data.ws** in the Data Files box of the main VuGen window.

The option to view a data file is available by default for Windows Sockets scripts. Note that you can only view the data in script view.



Several LRS functions, such as **lrs_receive** and **lrs_send**, handle the actual data that is transferred between servers and clients. The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the Vuser script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file, **data.ws**, contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRS functions use the buffer descriptors to access the data.

The descriptors have one of the following formats:

```
recv buf<i>index</i>   <i>number of bytes received</i>
send buf<i>index</i>
```

The buffer index begins with 0 (zero), and all subsequent buffers are numbered sequentially (1,2,3…) regardless of whether they are send or receive buffers.

In the following example, an **lrs_receive** function was recorded during a Vuser session:

```
lrs_receive("socket1", "buf4", LrsLastArg)
```

In this example, **lrs_receive** handled data that was received on socket1. The data was stored in the fifth receive record(buf4)—note that the index number is zero-based. The corresponding section of the **data.ws** file shows the buffer and its contents.

```
recv buf4 39
 "\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
 "\r\n"
 "\r\n"
 "SunOS UNIX (sunny)\r\n"
 "\r"
 "\x0"
 "\r\n"
 "\r"
 "\x0"
```

Understanding the Data File Format

The **data.ws** data file has the following format:

- File header
- A list of buffers and their contents

The file header includes an internal version number of the data file format. The current version is 2. If you try to access data from a data file with format version 1, VuGen issues an error.

```
;WSRData 2 1
```

An identifier precedes each record, indicating whether the data was received or sent, followed by the buffer descriptor, and the number of bytes received (for **lrs_receive** only). The buffer descriptor contains a number identifying the buffer.

For example,

```
recv buf5   25
```

indicates that the buffer contains data that was received. The record number is 5, indicating that this receive operation was the sixth data transfer (the index is zero based), and twenty-five bytes of data were received.

If your data is in ASCII format, the descriptor is followed by the actual ASCII data that was transferred by the sockets.

If your data is in EBCDIC format, it must be translated through a look-up table. For information on setting the translation table, see "Setting the WinSock Recording Options" on page 604. The EBCDIC whose ASCII value (after translation) is printable, is displayed as an ASCII character. If the ASCII value corresponds to a non-printable character, then VuGen displays the original EBCDIC value.

```
recv buf6 39
 "\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
 "\r\n"
 "SunOS UNIX (sunny)\r\n"
```

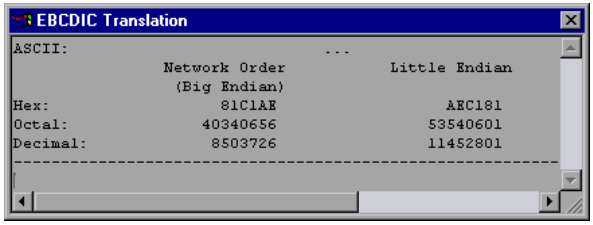
The following segment shows the header, descriptors, and data in a typical data file:

```
;WSRData 2 1

send buf0
 "\xff\xfd\x01\xff\xfd\x03\xff\xfb\x03\xff\xfb\x18"

recv buf1 15
 "\xff\xfd\x18\xff\xfd\x1f\xff\xfd"
 "#"
 "\xff\xfd"
 """
 "\xff\xfd"
 "$"

send buf2
 "\xff\xfb\x18"
```

Viewing Buffer Data in Hexadecimal format

VuGen contains a utility allowing you to view a segment of data, displaying it in hexadecimal and ASCII format, while indicating the offset of the data.

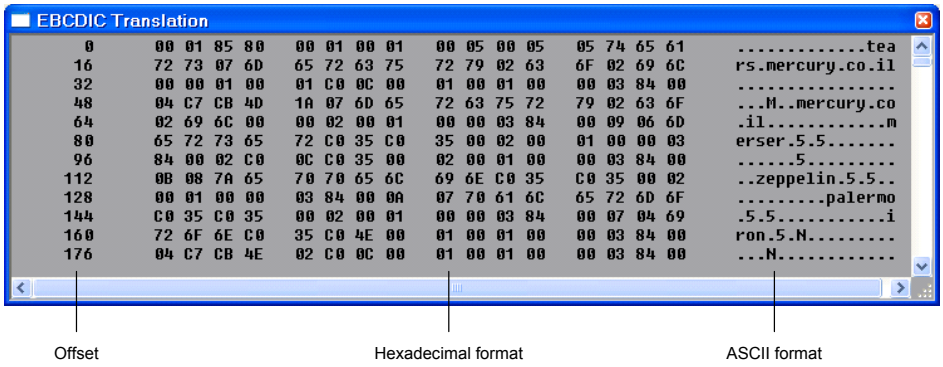
To display the data in the viewer window, select the data and press F7. If the selected text is less than four characters, VuGen displays the data in **short format**, showing the hexadecimal, decimal and octal representations.



You can customize the short format in the **conv_frm.dat** file as described in "Setting the Display Format" on page 628.

If the selected text is more than four characters, VuGen displays the data in several columns in **long format**. You can customize the long format by modifying the **conv_frm.dat** file, as described in "Setting the Display Format" on page 628.

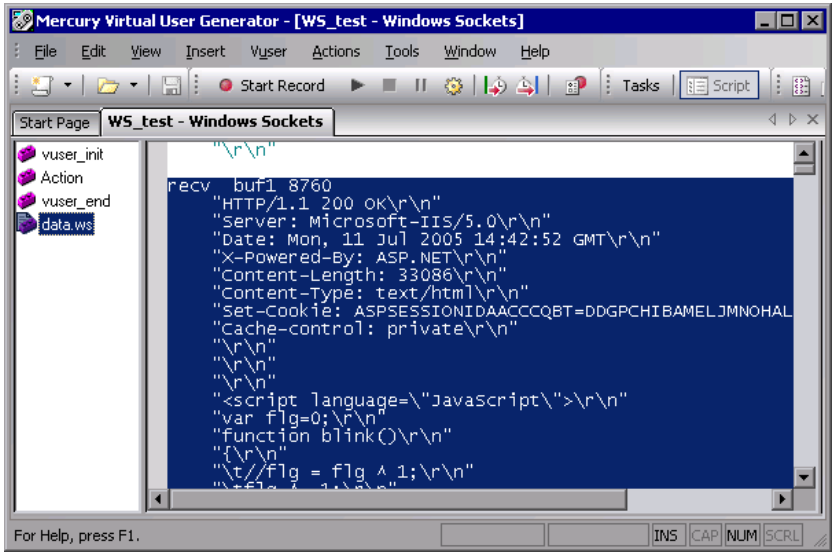
In the default format, the first column displays the character offsets from the beginning of the marked section. The second column displays the hexadecimal representation of the data. The third column shows the data in ASCII format. When displaying EBCDIC data, all non-printable ASCII characters (such as /n), are represented by dots.



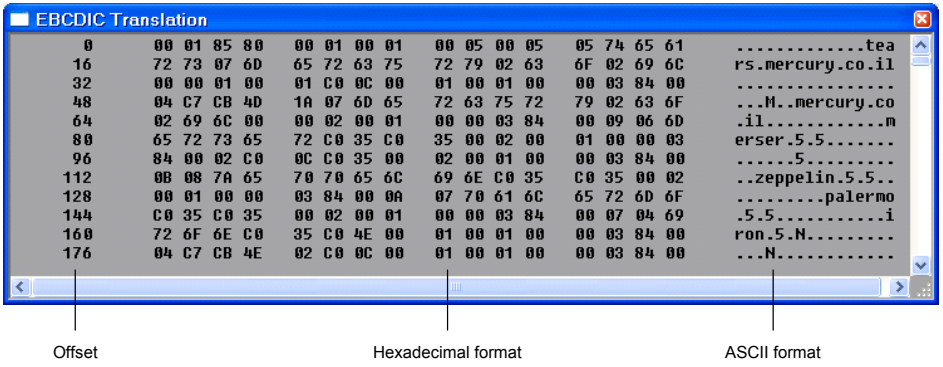
The F7 viewer utility is especially useful for parameterization. It allows you to determine the offset of the data that you want to save to a parameter.

To determine the offset of a specific character:

1 View **data.ws** and select the data from the beginning of the buffer.



2 Press F7 to display the data and the character offsets. Since more than four characters were selected, the data is displayed in long format.



3 Locate the value you want to correlate in the ASCII data. In this example, we will correlate the number 13546 (a process ID during a UNIX session) which begins at the 31st character—the last character in the second line.

4 Use the offset value in the **lrs_save_param_ex** function in order to correlate the value of the process ID. For more information, see Chapter 8, "Correlating Statements."

Setting the Display Format

You can specify how VuGen will display the buffer data in the viewer (F7) window. The **conv_frm.dat** file in the **lrun/dat** directory contains the following display parameters:

- **LongBufferFormat.** The format used to display five or more characters. Use nn for offset, XX for the hex data, and aa for ASCII data.
- **LongBufferHeader.** A header to precede each buffer in Long buffer format.
- **LongBufferFooter.** A footer to follow each buffer in Long buffer format.
- **ShortBufferFormat.** The format used to display four characters or less. You can use standard escape sequences and conversion characters.

The supported escape sequence characters are:

\a	**Bell (alert)**
\b	Backspace
\f	Formfeed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\?	Literal question mark
\ooo	ASCII character -octal

The supported conversion characters are:

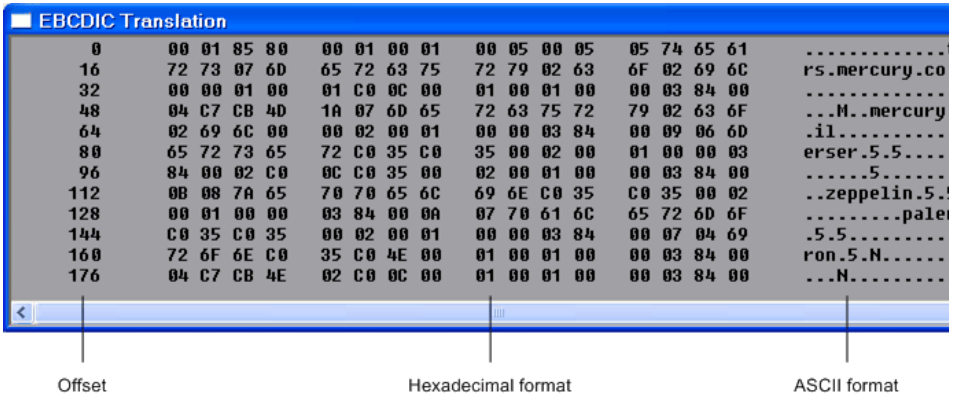
%a	ASCII representation
%BX	Big Endian (Network Order) Hex
%BO	Big Endian (Network Order) Octal
%BD	Big Endian (Network Order) Decimal
%LX	Little Endian Hex
%LO	Little Endian Octal
%LD	Little Endian Decimal

- **AnyBufferHeader.** A header to precede each buffer.
- **AnyBufferFooter.** A footer to follow each buffer.
- **NonPrintableChar.** The character with which to represent non-printable ASCII characters.
- **PrintAllAscii.** Set to 1 to force the printing of non-printable ASCII characters.

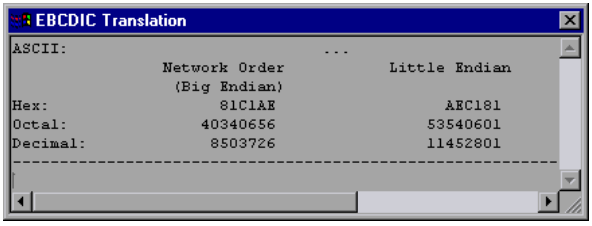
In the default settings, long and short formats are set, and a dot is specified for non-printable characters.

```
[BufferFormats]
LongBufferFormat=nnnnnnnn     XX XX XX XX   XX XX XX XX   XX XX XX XX   XX XX
XX XX     aaaaaaaaaaaaaaaa\r\n
LongBufferHeader=
LongBufferFooter=
ShortBufferFormat=ASCII:\t\t\t%a\r\n\t\tNetwork Order\t\tLittle Endian\r\n\t\t (Big
Endian)\r\nHex:\t\t%BX\t\t%LX\r\nOctal:\t\t%BO\t\t%LO\r\nDecimal:\t%BD\t\t%LD\r\n
AnyBufferHeader=
AnyBufferFooter=----------------------------------------------------------------\r\n
NonPrintableChar=.
PrintAllAscii=0
```

The default LongBufferFormat is displayed as:



The default ShortBufferFormat is displayed as:



Debugging Tips

VuGen offers several means which allow you to debug your script. You can view the various output logs and windows for detailed messages issued during execution.

Specifically for Windows Sockets Vuser scripts, VuGen provides additional information about buffer mismatches. A buffer mismatch indicates a variation in the received buffer size (generated during replay) and the expected buffer (generated during record). However, if the received and expected buffer are the same size, even though the contents are different, a mismatch message is not issued. This information can help you locate a problem within your system, or with your Vuser script.

You can view the buffer mismatch information in the Execution log. Select **View** > **Output** to display the Execution log if it is not visible.

Note that a buffer mismatch may not always indicate a problem. For example, if a buffer contains insignificant data such as previous login times, this type of mismatch can be ignored.

```
Mismatch (expected 54 bytes, 58 bytes actually received)
The expected buffer is:
=================
\r\n Last login: Wed Sep 2 10:30:18 from acme.hplab.c\r\n
=================
The received buffer is:
=================
\r\n Last login: Thu Sep 10 11:19:50 from dolphin.hplab.c\r\n
```

However, if there is a very large discrepancy between the size of the Expected and Received buffers, this could indicate a problem with your system. Check the data in the corresponding buffer for discrepancies.

In order for you to determine whether or not the mismatch is significant, you must thoroughly understand your application.

Manually Correlating WinSock Scripts

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

You can manually correlate a Vuser script using the correlation functions that save the dynamic values to a parameter. The **lrs_save_param and lrs_save_param_ex** functions let you save data to a parameter based on the offset of the data in the received buffer. An advanced correlation function **lrs_save_searched_string** lets you designate the data by specifying its boundaries and indicating which occurrence of the matched pattern to save to a parameter. The following example describes correlation using **lrs_save_param_ex.** For information about using other correlation functions, see the *Online Function Reference*.

To correlate the WinSock Vuser statements:

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

```
lrs_save_param_ex (socket, type, buffer, offset, length, encoding, parameter);
```

2 Reference the parameter.

View the buffer contents by selecting the **data.ws** file in the Data Files box of the main VuGen window. Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in angle brackets (< >).

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

```
frodo:/u/jay>ps
  PID TTY      TIME CMD
14602 pts/18    0:00 clock
14569 pts/18    0:03 tcsh

frodo:/u/jay>kill 14602
[3]    Exit 1                clock
frodo:/u/jay>
```

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

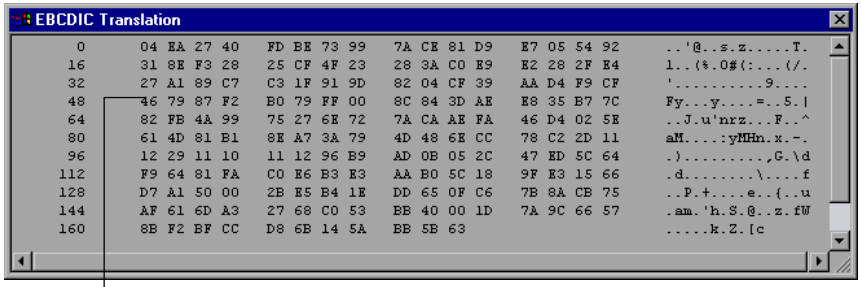
3 In the data.ws file, determine the buffer in which the data was received, buf47.

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

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

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

5 Determine the offset and length of the data string to save. Highlight the entire buffer and press F7. The offset of the **PID** is 11 and its length is 5 bytes. For additional information about displaying the data, see "Understanding the Data File Format" on page 623.



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

```
lrs_receive("socket1", "buf79", LrsLastArg);
lrs_save_param("socket1", "user", buf47, 11, 5, ascii, param1);
lr_output_message ("param1: %s", lr_eval_string("<param1>"));
lr_think_time(10);
lrs_send("socket1", "buf80", LrsLastArg);
```

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

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

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

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

37

Programming a Script in the VuGen Editor

In addition to recording a session, you can create a custom Vuser script. You can use both Vuser API functions and standard C, Java, VB, VBScript, or Javascript code.

This chapter includes:

About Creating Custom Vuser Scripts

VuGen allows you to program your own functions into the script, instead of recording an actual session. You can use the Vuser API or standard programming functions. Vuser API functions allow you to gather information about Vusers. For example, you can use Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test or monitoring.

This chapter describes how to program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes.

You can also develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API function libraries.

To create a customized script, you first create a skeleton script. The skeleton script contains the three primary sections of a script: **init**, **actions**, and **end**. These sections are empty and you manually insert functions into them.

You can create empty scripts for the following programming languages:

- C
- Java
- Visual Basic
- VBScript
- JavaScript

  Note: When working with JavaScript and VBScript Vusers, the COM objects that you use within your script must be fully automation compliant. This makes it possible for one application to manipulate objects in another application, or to expose objects so that they may be manipulated.

C Vusers

In C Vuser Scripts, you can place any C code that conforms with the standard ANSI conventions. To create an empty C Vuser script, select C Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty script:

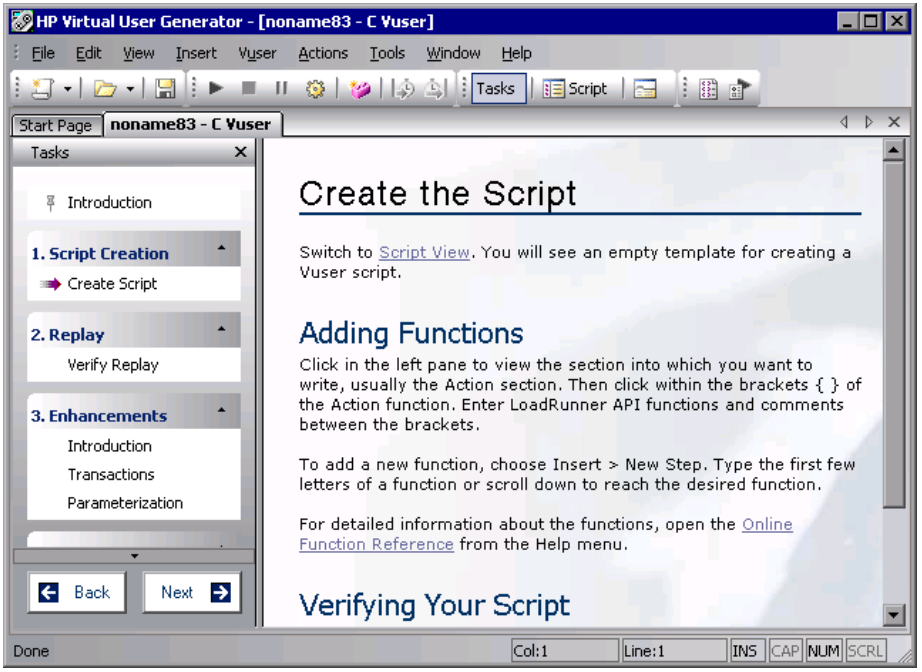
```
Action1()
{

    return 0;
}
```

You can use C Vuser functions in all of Vuser script types that use C functions.

You can also see the *Online Function Reference* (**Help > Function Reference**) for a C reference with syntax and examples of commonly used C functions.

Using the Workflow Wizard for C Vuser Scripts

The Workflow Wizard guides you through the steps of creating a script. By clicking on a link in the Tasks pane, you can read about the steps in creating a script, and view information about your replay. Use the **Back** and **Next** buttons to navigate between screens.



If you do not see the Workflow Wizard, make sure that the Tasks pane is open. (You show and hide the Task pane using the **Tasks** button on the toolbar). Then click the first link, **Introduction**.

See Chapter 4, "Viewing the VuGen Workflow" for more information about the wizard.

Create the Script

The Create Script window contains several guidelines for creating a Web Services script.

➤ **Adding Functions.** describes how and where to enter the functions.

➤ **Verifying Your Script.** describes how to verify your script after adding functions.

Guidelines for Using C Functions

All standard ANSI-C conventions apply to C Vuser scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other C programs. You declare and define variables using ANSI C conventions.

The C interpreter that is used to run Vuser scripts accepts the standard ANSI C language. It does not support any Microsoft extensions to ANSI C.

Before you add any C functions to a Vuser script, note the following limitations:

- A Vuser script cannot pass the address of one of its functions as a callback to a library function.
- The **stdargs**, **longjmp**, and **alloca** functions are not supported in Vuser scripts.
- Vuser scripts do not support structure arguments or return types. Pointers to structures are supported.
- In Vuser scripts, string literals are read-only. Any attempt to write to a string literal generates an access violation.
- C Functions that do not return int, must be casted. For example, `extern char * strtok();`

Calling libc Functions

In a Vuser script, you can call **libc** functions. However, since the interpreter that is used to run Vuser scripts does not support any Microsoft extensions to ANSI C, you cannot use Microsoft's include files. You can either write your own prototypes, or ask HP Customer Support to send you ANSI-compatible include files containing prototypes for **libc** functions.

Linking Mode

The C interpreter that is used to run Vuser scripts uses a "lazy" linking mode in the sense that a function need not be defined at the start of a run, as long as the function is defined before it is used. For example:

```
lr_load_dll("mydll.dll");
    myfun();  /* defined in mydll.dll -- can be called directly,
                 immediately after myfun.dll is loaded. */
```

Java Vusers

In Java Vuser scripts, you can place any standard Java code. To create an empty Java Vuser script, select Java Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty Java script:

```
import lrapi.lr;

public class Actions
{

    public int init() {
        return 0;
    }


    public int action() {
        return 0;
    }


    public int end() {
        return 0;
    }
}
```

Note that for Java type Vusers, you can only edit the **Actions** class. Within the Actions class, there are three methods: **init**, **action**, and **end**. Place initialization code in the **init** method, business processes in the **action** method, and cleanup code in the **end** method.

VB Vusers

You can create an empty Visual Basic Vuser script, in which you can place Visual Basic code. This script type lets you incorporate your Visual Basic application into VuGen. To create an empty VB Vuser script, select VB Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty VB script:

```
Public Function Actions() As Long

    '"TO DO: Place your action code here



    Actions = lr.PASS
End Function
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a VB function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the TO DO comment.

An additional section that is viewable from VuGen, is the **global.vba** file, which contains the object and variable global declarations for Vusers and the VB application.

Replay Error with VB Vuser Scripts

If you are getting error number -25210 when trying to replay a VB Vuser script, you may have a problem with some of your DLL files.

Solution:

1. Open the **c:\Program Files\Common Files\Microsoft Shared\vba\vba6** directory.
2. Locate the **VBE6.dll** and **VBE6EXT.OLB** files.
3. Right click the files and click properties to see the version of each file.
4. If either the **VBE6.dll** or the **VBE6EXT.OLB** file versions are between 6.04.9972 and 6.05.1024, they both must be replaced. If neither file version is in this range, contact HP software support.

5 Replace the **VBE6.dll** file with version 6.04.9972 or 6.05.1024.

6 Replace the **VBE6EXT.OLB** file with version 6.04.9969 or 6.05.1024.

VBScript Vusers

You can create an empty VBScript Vuser script, in which you can place VBScript code. This script type lets you incorporate your VBScript application into VuGen. To create an empty VBScript Vuser script, select VB Script Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty VBScript Vuser script:

```
Public Function Actions()

    '"TO DO: Place your action code here



    Actions = lr.PASS
End Function
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a VBScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the TO DO comment.

An additional section that is viewable from VuGen, is the **global.vbs** file, which creates the objects for the Vuser API functions and VB Script. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
Set lr = CreateObject("LoadRunner.LrApi")
```

JavaScript Vusers

You can create an empty JavaScript Vuser script, in which to place JavaScript code. This script type lets you incorporate your existing JavaScript application into VuGen. To create an empty JavaScript Vuser script, select JavaScript Vuser from the Custom category, in the New Virtual User dialog box.

```
function Actions()
{
    //"TO DO: Place your business process/action code here



    return(lr.PASS);
}
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a JavaScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the TO DO comment.

An additional section that is viewable from VuGen, is the **global.js** file, which creates the objects for the Vuser API functions and the Javascript. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
var lr = new ActiveXObject("LoadRunner.LrApi")
```

38

Programming Java Scripts

VuGen supports Java type users on a protocol level. This chapter explains how to create a Java Vuser script by programming. For information on creating a Java Vuser script through recording, see the chapters describing the Java protocol.

This chapter includes:

About Programming Java Scripts

To prepare Vuser scripts using Java code, use the **Java** type Vusers. This Vuser type supports Java on a protocol level. The Vuser script is compiled by a Java compiler and supports all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes "//".

Chapter 25, "Java Protocols - Recording" explains how to create a script through recording using the **Java Record Replay** Vuser. To prepare a Java coded script through programming, see the following sections.

The first step in creating a Java compatible Vuser script, is to create a new Vuser script template of the type **Java Vuser**. Then, you program or paste the desired Java code into the script template. You can add Java Vuser functions to enhance the script and parameterize the arguments to use different values during iterations.

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

After you prepare a script, run it as a standalone test from VuGen. A Java compiler (Sun's javac), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Creating a Java Vuser

The first step in creating a Java-compatible Vuser script is creating a Java Vuser template.

To create a Java Vuser script:

1. Open VuGen.
2. Select **File** > **New** or click the **New** button. The New Virtual User dialog box opens.
3. Select **Custom** > **Java Vuser** from the Select Vuser type list, and click **OK**. VuGen displays a blank Java Vuser script.
4. Click the **Actions** section in the left frame to display the **Actions** class.

Editing a Java Vuser Script

After generating an empty template, you can insert the desired Java code. When working with this type of Vuser script, you place all your code in the Actions class. To view the Actions class, click **Actions** in the left pane. VuGen displays its contents in the right pane.

```
import lrapi.*;
public class Actions
{
    public int init() {
        return 0;
    }

    public int action() {
        return 0;
    }

    public int end() {
        return 0;
    }
}
```

The Actions class contains three methods: init, action, and end. The following table shows what to include in each method and when each method is executed.

Script method	Used to emulate...	Is executed when...
init	a login to a server	the Vuser is initialized (loaded)
action	client activity	the Vuser is in "Running" status
end	a log off procedure	the Vuser finishes or is stopped

Init Method

Place all the login procedures and one-time configuration settings in the init method. The init method is only executed once—when the Vuser begins running the script. The following sample init method initializes an applet.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;

// Public function: init
    public int init() throws Throwable {

      // Initialize Orb instance...
      MApplet mapplet = new MApplet("http://chaos/classes/", null);
      orb = org.omg.CORBA.ORB.init(mapplet, null);
...
```

Action Method

Place all Vuser actions in the action method. The action method is executed according to the number of iterations you set in the runtime settings. For more information on the iteration settings, see Chapter 79, "Configuring Run-Time Settings." The following sample action method retrieves and prints the Vuser ID.

```
 public int action() {
     lr.message("vuser: " + lr.get_vuser_id() + " xxx");
          return 0;
   }
```

End Method

In the **end** method, place the code you want the Vuser to execute at the end of the script, such as logging off from a server, cleaning up the environment, and so forth.

The end method is only executed once—when the Vuser finishes running the script. In the following example, the end method closes and prints the end message to the execution log.

```
public int end() {
      lr.message("End");
      return 0;
  }
```

Java Vuser API Functions

VuGen provides a specific Java API for Java Vuser scripts. These functions are all static methods of the lrapi.lr class.

The Java API functions are classified into several categories: Transaction, Command Line Parsing, Informational, String, Message, and Run-Time functions.

For further information about each of these functions, see the *Online Function Reference* (**Help > Function Reference**). Note that when you create a new Java Vuser script, the *import lrapi.** is already inserted into the script.

To use additional Java classes, import them at the beginning of the script as shown below.

Remember to add the classes directory or relevant jar file to the classpath. Make sure that the additional classes are thread-safe and scalable.

```
import java.io.*;
import lrapi.*;

public class Actions
{
...
}
```

Working with Java Vuser Functions

You can use Java Vuser functions to enhance your scripts by:

- Inserting Transactions
- Inserting Rendezvous Points
- Obtaining Vuser Information
- Issuing Output Messages
- Emulating User Think Time
- Handling Command Line Arguments

Inserting Transactions

You define transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified requests. These requests can be short or complex tasks. When working with LoadRunner, you can analyze the performance per transaction during and after the scenario run, using online monitor and graphs.

You can also specify a transaction status: lr.PASS or lr.FAIL. You can let the Vuser automatically determine if the transaction was successful, or you can incorporate it into a conditional loop. For example, in your code you can check for a specific return code. If the code is correct, you issue a lr.PASS status. If the code is wrong, you issue an lr.FAIL status.

To mark a transaction:

1. Insert **lr.start_transaction** into the script, at the point where you want to begin measuring the timing of a task.
2. Insert **lr.end_transaction** into the script, at the point where you want to stop measuring the task. Use the transaction name as it appears in the **lr.start_transaction** function.

3 Specify the desired status for the transaction: lr.PASS or lr.FAIL.

```
public int action() {

for(int i=0;i<10;i++)
  {
    lr.message("action()"+i);
    lr.start_transaction("trans1");
    lr.think_time(2);
    lr.end_transaction("trans1",lr.PASS);
  }
    return 0;
}
```

Inserting Rendezvous Points

The following section does not apply to the HP Business Availability Center.

To emulate heavy user load on your client/server system, you synchronize Vusers to perform a task at exactly the same moment by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it is held by the Controller until all Vusers participating in the rendezvous arrive.

You designate the meeting place by inserting a rendezvous function into your Vuser script.

To insert a rendezvous point:

➤ Insert an lr.rendezvous function into the script, at the point where you want the Vusers to perform a rendezvous.

```
public int action() {

 for(int i=0;i<10;i++)
  {
    lr.rendezvous("rendz1");
    lr.message("action()"+i);
    lr.think_time(2);
  }
  return 0;
}
```

Obtaining Vuser Information

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

lr.get_attrib_string	Returns a string containing command line argument values or runtime information such as the Vuser ID or the load generator name.
lr.get_group_name	Returns the name of the Vuser's group.
lr.get_host_name	Returns the name of the load generator executing the Vuser script.
lr.get_master_host_name	Returns the name of the machine running the LoadRunner Controller or Business Process Monitor.
lr.get_scenario_id	Returns the ID of the current scenario. (LoadRunner only)
lr.get_vuser_id	Returns the ID of the current Vuser. (LoadRunner only)

In the following example, the lr.get_host_name function retrieves the name of the computer on which the Vuser is running.

```
String my_host = lr.get_host_name();
```

For more information about the above functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Issuing Output Messages

When you run a scenario, the Controller Output window displays information about script execution. You can include statements in a Vuser script to send error and notification messages to the Controller. The Controller displays these messages in the Output window. For example, you could insert a message that displays the current state of the client application. You can also save these messages to a file.

Note: Do not send messages from within a transaction. Doing so lengthens the transaction execution time and may skew the actual transaction results.

You can use the following message functions in your Vuser script:

lr.debug_message	Sends a debug message to the Output window.
lr.log_message	Sends a message to the Vuser log file.
lr.message	Sends a message to a the Output window.
lr.output_message	Sends a message to the log file and Output window with location information.

In the following example, **lr.message** sends a message to the output indicating the loop number:

```
for(int i=0;i<10;i++)
  {
    lr.message("action()"+i);
    lr.think_time(2);
  }
```

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

You can instruct the Vusers to redirect the Java standard output and standard error streams to VuGen's Execution log. This is especially helpful when you need to paste existing Java code or use ready-made classes containing **System.out** and **System.err** calls in your Vuser scripts. In the execution log, standard output messages are colored blue, while standard errors are shown in red.

The following example shows how to redirect specific messages to the standard output and standard error using lr.enable_redirection:

```
lr.enable_redirection(true);

System.out.println("This is an informatory message…");  // Redirected
System.err.println("This is an error message…");  // Redirected

lr.enable_redirection(false);

System.out.println("This is an informatory message…");  // Not redirected
System.err.println("This is an error message…");  // Not redirected
```

Note: When you set **lr.enable_redirection** to **true**, it overrides all previous redirections. To restore the former redirections, set this function to **false**.

For additional information about this function, see the *Online Function Reference* (**Help** > **Function Reference**).

Emulating User Think Time

The time that a user waits between performing successive actions is known as the think time. Vusers use the lr.think_time function to emulate user think time. In the following example, the Vuser waits two seconds between loops:

```
for(int i=0;i<10;i++)
  {
    lr.message("action()"+i);
    lr.think_time(2);
  }
```

You can use the think time settings as they appear in the script, or a factor of these values. To configure how Vusers handle think time functions, open the runtime settings dialog box. For more information, see Chapter 79, "Configuring Run-Time Settings."

For more information about the **lr.think_time** function, see the *Online Function Reference* (**Help > Function Reference**).

Handling Command Line Arguments

You can pass values to a Vuser script at runtime by specifying command line arguments when you run the script. You insert command line options after the script path and filename in the Controller or Business Process Monitor. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

lr.get_attrib_double	Retrieves double precision floating point type arguments
lr.get_attrib_long	Retrieves long integer type arguments
lr.get_attrib_string	Retrieves character strings

Your command line should have 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 machine pc4:

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

For more information on the command line parsing functions, see the *Online Function Reference* (**Help > Function Reference**). For more information on how to insert the command line options, see the *LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Setting your Java Environment

Before running your Java Vuser script, make sure that the environment variables, PATH and CLASSPATH, are properly set on all machines running Vusers:

- To compile and replay the scripts, you must have complete JDK installation, either version 1.1 or 1.2, or 1.3. The installation of the JRE alone is not sufficient. It is preferable not to have more than one JDK or JRE installation on a machine. If possible, uninstall all unnecessary versions.
- The **PATH** environment variable must contain an entry for **JDK/bin**.
- For JDK 1.1.x, the **CLASSPATH** environment variable must include the **classes.zip** path, (**JDK/lib** subdirectory) and all of the VuGen classes (**classes** subdirectory).
- All classes used by the Java Vuser must be in the classpath—either set in the machine's **CLASSPATH** environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

Running Java Vuser Scripts

Java Vuser scripts differ from C Vuser scripts in that they are first compiled and then executed; C Vuser scripts are interpreted. VuGen locates the **javac** compiler from within the JDK installation and compiles the Java code inside the script. This stage is indicated by the **Compiling...** status message in the bottom of the VuGen window. If errors occur during compilation, they are listed in the execution log. To go to the code in your script that caused the error, double-click on the error message containing the line number of the error. Fix the error and run the script again.

If the compilation succeeds, the status message **Compiling...** changes to **Running...** and VuGen begins to execute the script. When you run the script again, VuGen runs the script without recompiling it, provided that no changes were made to the script. To debug your script further, you can use breakpoints and animated run type execution using the step option.

Note: If you are making calls to JNDI extensions within your script, you may encounter problems trying to run your Vusers as **threads**. This happens because JNDI requires each thread to have its own context class loader. In order to run as threads, instruct each Vuser to run with its own context class loader, by adding the following line to the beginning of the **init** section:

```
DummyClassLoader.setContextClassLoader();
```

Compiling and Running a Script as Part of a Package

When creating a Java Vuser script, you may need to use methods in other classes in which the class or method is protected. If you try to compile this type of script, you will receive errors in the compilation stage indicating that the methods are inaccessible. To make sure that your script can access these methods, insert the package name containing these methods at the top of the script, just as you would do in a standard Java program—<package_name>. In the following example, the script defines the just.do.it package which consists of a path:

```
package my.test;

import lrapi.*;
public class Actions
{
    :
}
```

In the above example, VuGen automatically creates the **my/test** directory hierarchy under the Vuser directory, and copies the **Actions.java** file to **my/test/Actions.java**, allowing it to compile with the relevant package. Note that the package statement must be the first line in the script, similar to Java (excluding comments).

Programming Tips

When programming a Java Vuser script, you can paste ready-made code segments into scripts or import ready-made classes in order to invoke their methods. If Vusers need to run as threads under the Controller (for scalability reasons), you need to make sure that all of the imported code is thread-safe.

Thread-safety is often difficult to detect. A Java Vuser may run flawlessly under VuGen and under the Controller with a limited number of Vusers. Problems occur with a large number of users. Code that is not thread-safe is usually the result of static class member usage as shown in the following example:

```
import lrapi.*;
public class Actions
{
   private static int iteration_counter = 0;

     public int init() {
          return 0;
     }

     public int action() {
          iteration_counter++;
          return 0;
     }

     public int end() {
          lr.message("Number of Vuser iterations: "+iteration_counter);
          return 0;
     }
}
```

When you run one Vuser, the **iteration_counter** member accurately determines the number of iterations that were executed. When multiple Vusers run together as threads on a single virtual machine, the static class member **iteration_counter** is shared by all threads, resulting in an incorrect counting. The total number of all Vusers iterations is counted.

If code is known to be non thread-safe and you still want to import it into your script, you can run the Vusers as processes. For more information on running Vusers as threads or processes, see Chapter 79, "Configuring Run-Time Settings."

When you run a basic Java Vuser script, it usually consists of a single thread—the main thread. Only the main thread can access the Java Vuser API. If a Java Vuser spawns secondary worker threads, using the Java API may cause unpredictable results. Therefore, we recommend that you use the Java Vuser API only in the main thread. Note that this limitation also affects the **lr.enable_redirection** function.

The following example illustrates where the LR API may and may not be used. The first log message in the execution log indicates that the value of flag is false. The virtual machine then spawns a new thread set_thread. This thread runs and sets flag to true, but will not issue a message to the log, even though the call to lr.message exists. The final log message indicates that the code inside the thread was executed and that flag was set to true.

```
boolean flag = false;

public int action() {
   lr.message("Flag value: "+flag);
   Thread set_thread = new Thread(new Runnable();{
         public void run() {
            lr.message("LR-API NOT working!");
            try {Thread.sleep(1000);} catch(Exception e) {}
            flag = true;
         }
   });
   set_thread.start();
   try {Thread.sleep(3000);} catch(Exception e) {}
   lr.message("Flag value: "+flag);
   return 0;
}
```

39

COM Protocol

Many Windows applications use COM-based functions either directly, or through library calls. You can use VuGen to record a script that emulates a COM-based client accessing a COM server. The resulting script is called a COM Vuser script. You can also create COM Vuser scripts by using a Visual Basic add-in. For more information about the Visual Basic add-in, see the Virtual User Guide appendix.

Chapter 40, "COM - Understanding and Correlating," explains how COM Vuser scripts operate.

This chapter includes:

About Recording COM Vuser Scripts

When you record COM client applications, VuGen generates functions that describe COM client-server activity. The recorded script contains interface declarations, API calls and instance calls to methods. Each COM function begins with an **lrc** prefix.

You can view and edit the recorded script from the VuGen's main window. The COM API/method calls that were recorded during the session are displayed in the window, allowing you to visually track application COM/DCOM calls.

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. See "Setting Script Generation Preferences" on page 1211 for more information.

COM Overview

This section provides an outline of COM technology. This should be enough to get you started with COM Vuser scripts. See Microsoft Developer's Network (MSDN) and other documentation for further details.

COM (Component Object Model) is a technology for developing reusable software components ("plug-ins"). DCOM (Distributed COM) allows use of COM components on remote computers. Microsoft transaction servers (MTS), Visual Basic and Explorer all use COM/DCOM technology. Thus, the application you are testing may use COM technology indirectly, even though you don't know it. You will probably have to include some, but certainly not all, of the COM calls made by your application in the Vuser script.

Objects, Interfaces and Type Libraries

COM objects are binary code modules. Each COM object implements one or more interfaces that allow client programs to communicate with it. You need to know about these interfaces in order to follow the COM calls in the Vuser scripts. Type libraries, used as a reference for accessing COM interface methods and parameters, contain descriptions of COM objects and interfaces. Each COM class, interface, and type library is identified by a Global Unique Identifier (GUID).

COM Interfaces

A COM interface provides a grouped collection of related methods. For example, a **Clock** object may have **Clock**, **Alarm** and **Timer** interfaces. Each interface has one or more methods. For example the **Alarm** interface may have **AlarmOn** and **AlarmOff** methods.

An interface may also have one or more properties. Sometimes, the same function may be performed by calling a method or by setting or getting the value of a property. For example, you can set the **Alarm Status** property to **On** or call the **AlarmOn** method.

A COM object may support many interfaces. The **IUnknown** interface is implemented by all components and is used to find out about other interfaces. Many components also implement the **IDispatch** interface, which exposes all other interfaces and methods of the object, allowing implementation of COM automation in scripting languages.

COM Class Context and Location Transparency

COM objects can run on the same machine as the client application, or on a remote server. COM objects that an application creates may be in a local library, a local process or a remote machine ("Remote Object Proxy"). The location of the COM object, known as the "Context," can be transparent to the application. Most users apply the Vusers to check the load on remote servers. Therefore, objects accessed by Remote Object Proxy are usually the most relevant for these tests.

COM Data Types

COM also provides several special data types, including safe arrays, BSTR strings and variants. You may need to use these data types for debugging, parameterization and similar tasks.

Getting Started with COM Vusers

This section describes the process of developing COM Vuser scripts.

To develop a COM Vuser script:

1 Record the basic script using VuGen.

Start VuGen and create a new Vuser script. Specify COM as the type of Vuser. Select an application to record and set the recording options. To set the script related recording options, see Chapter 76, "Setting Script Generation Preferences." To set the COM specific options and filters, see the "Setting COM Recording Options" on page 670. Record typical operations using your application.

For details about recording, see Chapter 5, "Recording with VuGen."

2 Refine the Object Filter.

Use the log file that was generated to refine your choice of objects to be recorded in the filter. See the following section, "Selecting COM Objects to Record", for details.

3 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Selecting COM Objects to Record

The application you are testing may use a great many COM objects. Only a few may actually create load and may be important for the load test. Thus, before you record a COM application, you should select the objects you want to record for the load test. VuGen allows you to browse for objects from type libraries that it can read on the local machine and on other computers in the network.

Deciding Which Objects to Use

There are several ways to decide which COM objects should be included in the test. Try to determine which remote objects are used by the software. If you are unsure which objects to use, try using the default filter. The Environments branch of the filter includes calls to three sets of objects (ADO, RDS and Remote) that are likely to generate load on remote servers.

You can also check the actual calls to refine the filter. After you have recorded the test, you can save the file and look in the **data** directory that VuGen creates for a file named **lrc_debug_list_<nnn>.log**", where **nnn** is the process number. This log file contains a listing of each COM object that was called by the recorded application, regardless of whether or not the recording filter included that object. Only calls that generate load on the server should be included for recording.

For example, the following is a local COM of the Visual Basic library:

```
Class JetES {039EA4C0-E696-11D0-878A-00A0C91EC756}
was loaded from type library "JET Expression Service Type Library"
({2358C810-62BA-11D1-B3DB-00600832C573} ver 4.0)
```

It should not be added since it does not generate load on the server.

Likewise, since the OLE DB and Microsoft Windows Common Controls are local objects, the following are examples of classes and libraries that are not going to place any load on the server and should not be recorded:

```
Class DataLinks {2206CDB2-19C1-11D1-89E0-00C04FD7A829}
was loaded from type library "Microsoft OLE DB Service Component 1.0 Type Library"
({2206CEB0-19C1-11D1-89E0-00C04FD7A829} ver 1.0)

Class DataObject {2334D2B2-713E-11CF-8AE5-00AA00C00905}
was loaded from type library "Microsoft Windows Common Controls 6.0 (SP3)"
({831FDD16-0C5C-11D2-A9FC-0000F8754DA1} ver 2.0)
```

However, for example, a listing such as the following indicates a class that should be recorded since it does generate load on the server:

```
Class Order {B4CC7A90-1067-11D4-9939-00105ACECF9A}
was loaded from type library "FRS"
({B4CC7A8C-1067-11D4-9939-00105ACECF9A} ver 1.0)
```

Calls to classes of the **FRS** library, used for instance in the flight_sample that is installed with VuGen, use server capacity and should be recorded.

If a COM object itself calls other COM objects, all the calls will be listed in the type information log file. For example, every time the application calls an **FRS** class function, the **FRS** library calls the **ActiveX Data Object (ADO)** library. If several functions in such a chain are listed in a filter, VuGen records only the first call that initiates the chain. If you selected both **FRS** and **ADO** calls, only the **FRS** calls will be recorded.

On the other hand, if you select only the **ADO** library in the filter, then calls to the **ADO** library will be recorded. It is often easier to record the call to the first remote object in the chain. In some cases, however, an application may use methods from several different COM objects. If all of them use a single object that puts a load on the server, you could only record the final common object.

Which Objects Can Be Selected

VuGen can only record objects if it can read their type libraries. If the type libraries were not installed in the system or VuGen cannot find them, the COM objects will not be listed in the Recording Options dialog box. If they are used by your application, VuGen will not be able to identify these objects and will identify them as **INoTypeInfo** in the files.

Which Interfaces Can Be Excluded

For each object, the Recording Options dialog box will show you all interfaces that are listed in the Type Library, and allow you to specify inclusion or exclusion of each one. However, **ADO**, **RDS** and **Remote Objects** can be included in the filter as a group. The filter will not show the individual objects of those environments or their interfaces. Objects that you included from type libraries may also have interfaces that are not listed in the type library and therefore not shown in the Recording Options dialog. After generating a VuGen script, you can identify these interfaces in the script and get their GUID numbers from the interfaces.h file that VuGen generates. Using this information, you can exclude the interfaces as explained below.

Setting COM Recording Options

Use the COM Recording Options dialog box to set the filtering and COM scripting options. You use the online browser to locate type libraries in the registry, file system, or the Microsoft Transaction Server (MTS).

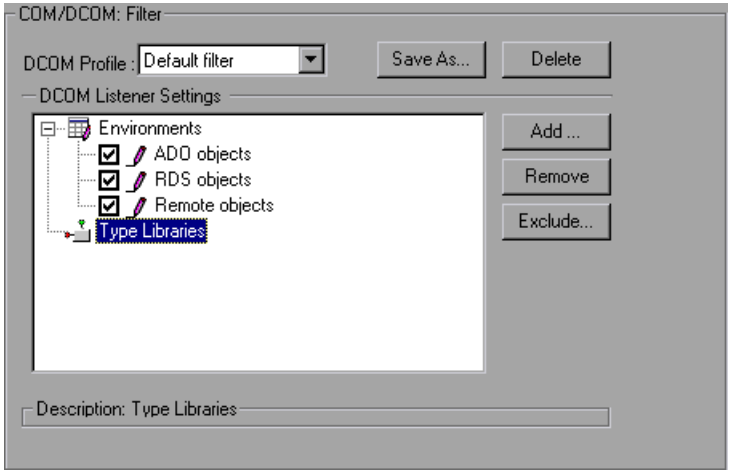
For more information, see:

- Filtering Objects
- Setting the Filter
- Setting COM Scripting Options

Filtering Objects

The Filter options let you indicate which COM objects should be recorded by VuGen. You can select objects from within environments and libraries.

The Filter options set a default filter or create alternate filters. You can filter a recording session by environment and type libraries.



DCOM Profile

- **Default Filter.** The filter to be used as the default when recording a COM Vuser script.
- **New Filter.** A clean filter based on the default environment settings. Note that you must specify a name for this filter before you can record with its settings.

DCOM Listener Settings

The DCOM Listener Settings display a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, a dialog box opens asking you if you also want to exclude the interface in all classes that implement it this interface.

Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

- **Environment.** The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record.
- **Type Libraries.** A type library **.tlb** or **.dll** file, that represents the COM object to record. All COM objects have a type library that represents them. You can select a type library from the Registry, Microsoft Transaction Server, or file system.

Type Libraries. In the lower section of the dialog box, VuGen displays the following information for each type library.

- **TypLib.** The name of the type library (tlb file).
- **Path.** The path of the type library.
- **Guid.** The Global Unique Identifier of the type library.

Setting the Filter

This section describes how to set the filters.

To select which COM objects to record:

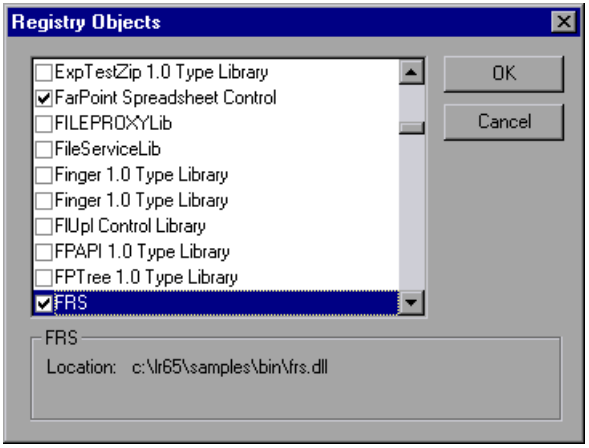
1 Select **Tools** > **Recording Options** from the main menu or click **Options** in the Start Recording dialog box. A dialog box opens displaying the Recording Options tree. Select the **COM/DCOM:Filter** node.

Expand the Environments sub-tree, to display the **ADO**, **RDS** and **Remote objects** listings. The Filter also includes a **Type Libraries** tree that is initially empty. You can add Type Libraries as described in the steps below.

By default, all Environments are selected and calls to any of their objects are included in the filter. Clear the check box adjacent to **ADO**, **RDS** or **Remote objects** to exclude them from the filter.

2 Click **Add** to add another COM type library, and select a source to browse: registry, file system, or MTS, as described below.

3 Select **Browse Registry** to display a list of type libraries found in the registry of the local computer.



Select the check box next to the desired library or libraries and click **OK**.

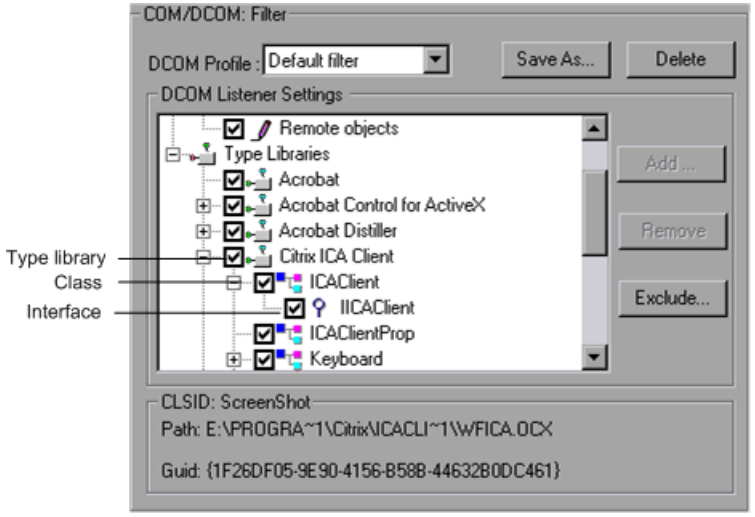
4 To add a type library from the file system, click **Add** and select **Browse file system**.

Select the desired file and click **OK**.

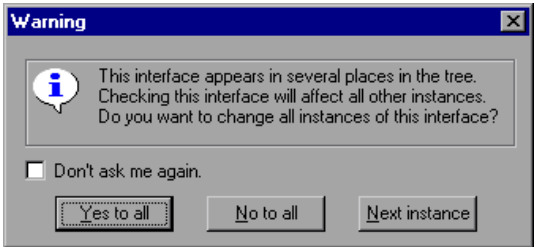
5 Once the type library appears in the list of Type Libraries, you can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

When you clear a check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

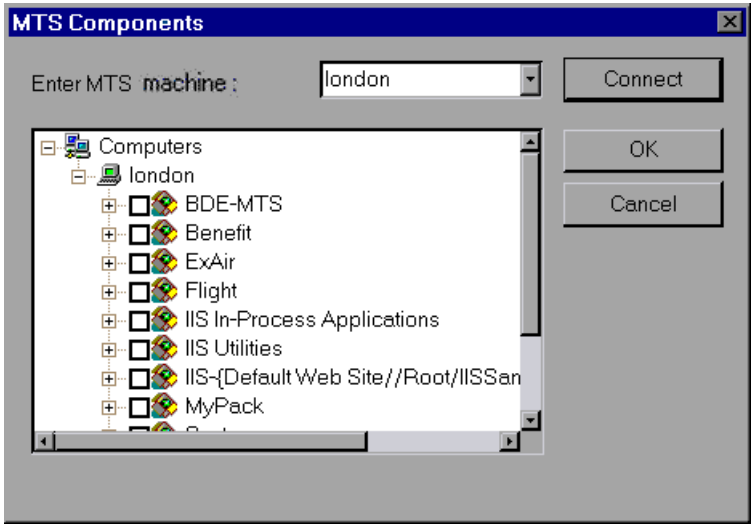


6 An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the following warning:



If you check **Don't ask me again** and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click **Yes to all** to change the status of all instances of this interface for all other classes, click **No to all** to leave the status of all other instances unchanged. Click **Next Instance** to view the next class that uses this interface.

7 To add a component from a Microsoft Transaction Server, click **Add** and select **Browse MTS**. The MTS Components dialog box prompts you to enter the name of the MTS server.

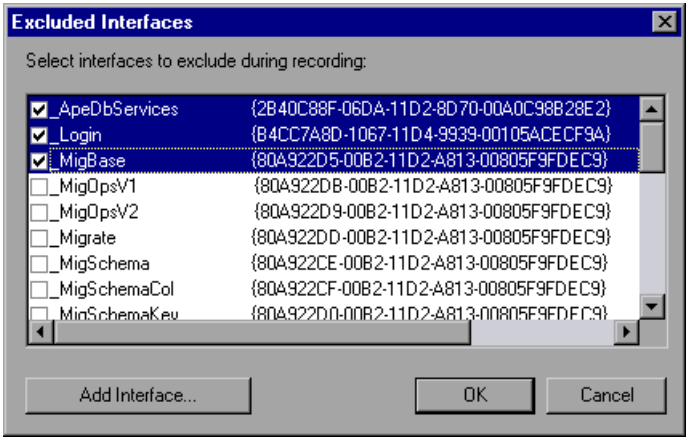


Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine.

Select one or more packages of MTS components from the list of available packages and click **Add**. Once the package appears in the list of Type Libraries, you can select specific components from the package.

8 In addition to disabling and enabling recording of interfaces in the tree display, you can also click **Exclude** in the Recording Options dialog to include or exclude interfaces in the filter, whatever their origin.

Note that you can also exclude classes and interfaces by clearing the check box adjacent to the item, inside the type library tree hierarchy.



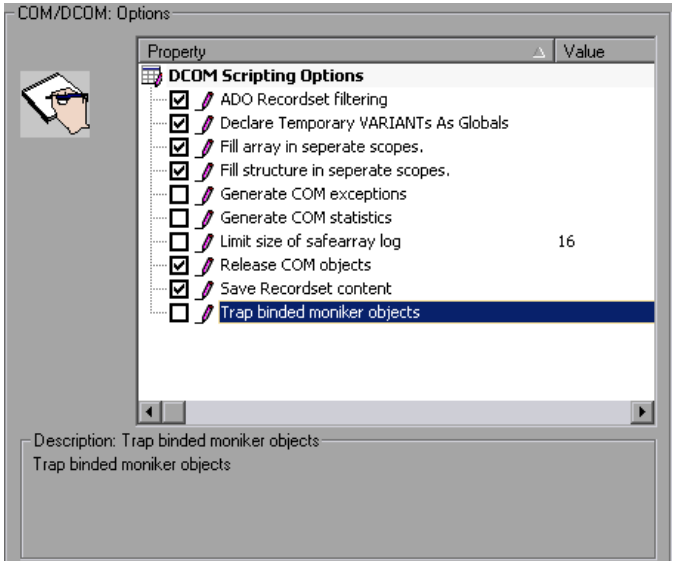
The checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click **Add Interface...** in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the **Add Interface...** feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter.

9 When you have finished modifying your filter, click **OK** to save it and close the dialog box. Click **Save As** to save a New filter, or to save an existing filter under a new name. You can select saved filters in subsequent recordings. Default settings are given initially in the **Default filter**.

Setting COM Scripting Options

You can set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.



- **ADO Recordset filtering.** Condense multiple recordset operations into a single-line fetch statement (enabled by default).
- **Declare Temporary VARIANTs as Globals.** Define temporary VARIANT types as Globals, not as local variables (enabled by default).
- **Fill array in separate scopes.** Fill in each array in a separate scope (enabled by default).
- **Fill structure in separate scopes.** Fill in each structure in a separate scope (enabled by default).
- **Generate COM exceptions.** Generate COM functions and methods that raised exceptions during recording (disabled by default).
- **Generate COM statistics.** Generate recording time performance statistics and summary information (disabled by default).

- **Limit size of SafeArray log.** Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).
- **Release COM Objects.** Record the releasing of COM objects when they are no longer in use (enabled by default).
- **Save Recordset content.** Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).
- **Trap binded moniker objects.** Trap all of the bound moniker objects (disabled by default).

To set COM/DCOM options:

1. Select **Tools** > **Recording Options** from the main menu or click **Options** in the Start Recording dialog box. VuGen opens the Recording Options tree. Select the **COM/DCOM:Options** node.
2. Enable the desired options by clicking the check boxes adjacent to them.
3. Click **OK** to save your settings and exit.

40

COM - Understanding and Correlating

This chapter provides details about the scripts VuGen generates for COM client communications, including an explanation of the function calls and examples. For basic information about getting started with COM Vuser scripts, see Chapter 39, "COM Protocol."

This chapter includes:

About COM Vuser Scripts

For each COM Vuser script, VuGen creates the following:

- Interface pointer and other variable declarations in file interfaces.h
- Function calls that you can record in the vuser_init, actions or vuser_end sections.
- A user.h file containing the translation of the Vuser script into low level calls

When you record COM client communications, VuGen creates a script with calls to COM API functions and interface methods. In addition, you can manually program COM type conversion functions.

After you record the script, you can view any of these files by selecting them from the tree on the left-hand side of the VuGen screen.

Each VuGen COM function has an **lrc** prefix, such as **lrc_CoCreateInstance** or **lrc_long**. The lrc functions are classified into several categories: Creating Instances, IDispatch Interface Calls, Type Conversion from String, Assignment to Variants, Create New Variants, Parameterization, Extracting From Variants. Array Types, ADO Recordset, Byte Array, VB Collection Support, and Debug functions.

For syntax and examples of the lrc functions, see the *Online Function Reference* (**Help > Function Reference**).

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see Chapter 76, "Setting Script Generation Preferences."

Understanding VuGen COM Script Structure

VuGen COM scripts are structured in a special way to meet the needs of COM interfaces.

Interface Methods

Calls to interface methods have the following names and syntax conventions:

```
lrc_<interface name>_<method name>(instance,...);
```

Note that the **instance** is always the first parameter passed.

The vendors of the respective COM components usually supply documentation for the interface functions.

Interface Pointers

The interface header file defines the interface pointers, as well as other variables, that can be used in the script. Each interface has an Interface ID (IID) which uniquely identifies the interface.

The format of the interface definition is:

```
<interface type>*<interface name> = 0; //"{<IID of the interface type>}"
```

In the following example, the interface type is IDispatch, the name of the interface instance is IDispatch_0, and the IID of IDispatch type is the long number string:

```
IDispatch* IDispatch_0= 0;//"{00020400-0000-0000-C000-000000000046}"
```

Vuser Script Statements

The COM Vuser script consist of code that creates object instances, retrieves interface pointers and calls the interface methods. Each user action may generate one or more COM calls. Each COM call is coded by VuGen as a group of statements. Each such group is contained in a separate scope enclosed in braces. Several different statements prepare for the main call by assigning values and performing type conversions. For example, the group of calls needed to create an object may look like this:

```
{
GUID pClsid = lrc_GUID("student.student.1");
IUnknown * pUnkOuter = (IUnknown*)NULL;
unsigned long dwClsContext = lrc_ulong("7");
GUID riid = IID_IUnknown;
lrc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid, (void**)&IUnknown_0,
CHECK_HRES);
}
```

Error Checking

Each COM method or API call returns an error value. VuGen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

CHECK_HRES	This value is inserted if the function passed during recording and errors should be checked during replay.
DONT_CHECK_HRES	This value is inserted if the function failed during recording and errors should not be checked during replay.

Examining Sample VuGen COM Scripts

This section shows examples of how VuGen emulates a COM client application.

Note that VuGen displays the results of a query in a grid. You can view up to 200 records by scrolling through the grid. For more information, see "Working with Grids" on page 586

Basic COM Script Operations

The basic operations are:

- Instantiation of the object
- Retrieving interface pointers
- Calling interface methods

Each type of operation is done within a separate scope.

Instantiation of the Object

To use a COM object, the application must first instantiate it and get a pointer to an interface of that object.

VuGen does the following to instantiate an object:

1 VuGen calls lrc_GUID to get a unique ProgID for the object, to be stored in pClsid:

```
GUID pClsid = lrc_GUID("student.student.1");
```

pClsid is the unique global CLSID of the object, which was converted from the ProgID **student.student.1**

2 If the unknown interface pointer is a pointer to an aggregated object, VuGen retrieves the pointer to that object, or else it sets it to NULL:

```
IUnknown * pUnkOuter = (IUnknown*)NULL;
```

3 VuGen sets the contexts of the object to be created:

```
unsigned long dwClsContext = lrc_ulong("7");
```

dwClsContext contains the context of the object (in process, local, remote or combinations of these.)

4 VuGen sets a variable to hold the requested interface ID, which is IUnknown in this case:

```
GUID riid = IID_IUnknown;
```

riid contains the interface ID of the **IUnknown** interface.

5 After the input parameters are prepared, a call to lrc_CoCreateInstance creates an object using the parameters defined in the preceding statements. A pointer to the IUnknown interface is assigned to output parameter IUnknown_0. This pointer is needed for subsequent calls:

```
lrc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid, (void**)&IUnknown_0,
CHECK_HRES);
```

The input parameters were prepared and explained above. Since the call succeeded, VuGen sets error checking on during the user simulation by inserting the **CHECK_HRES** value. The call returns a pointer to the IUnknown interface in **IUnknown_0**, that can be used in subsequent calls.

Retrieving an Interface

After creating an object, VuGen has access only to the **IUnknown** interface. VuGen will use the **IUnknown** interface for communicating with the object. This is done using the **QueryInterface** method of the **IUnknown** standard interface. The first parameter in a VuGen method call is the interface instance. In this case it is the **IUnknown_0** pointer set previously by **CoCreateInstance**. The **QueryInterface** call requires as input the ID of the interface to be retrieved, and returns a pointer to the interface designated by that ID.

To get the interface:

1 First, VuGen sets a parameter, riid, equal to the ID of the Istudent interface:

```
GUID riid = IID_Istudent;
```

2 A call to QueryInterface assigns a pointer to the Istudent interface to output parameter Istudent_0 if the Istudent object has such an interface:

```
lrc_IUnknown_QueryInterface(IUnknown_0, &riid, (void**)&Istudent_0,
CHECK_HRES);
```

Using an Interface to Set Data

Here is an example of using the methods of the interface to set data. Suppose that in the application, the user is supposed to input a name. This activates a method for setting the name. VuGen records this in two statements. One statement is used for setting up the name string and the second one sets the name property.

To set up the entire function call:

1 First, VuGen sets a variable (Prop Value) equal to the string. The parameter is of type BSTR, a string type used in COM files:

```
BSTR PropValue = lrc_BSTR("John Smith");
```

In subsequent stages, you will probably parameterize this call, replacing "John Smith" with a parameter, so that different names are used each time the Vuser script is run.

2 Next, VuGen calls the Put_Name method of the Istudent interface to enter the name:

```
lrc_Istudent_put_name(Istudent_0, PropValue, CHECK_HRES);
```

Using an Interface to Return Data

Returning data from an application is different than entering the data, because you might want to store these values and use them as inputs in subsequent calls for parameterization.

The following is an example of what VuGen may do when the application retrieves data:

1 Create a variable of the appropriate type (in this case a BSTR) that will contain the value of the property.

```
BSTR pVal;
```

2 Get the value of the property, in this case a name, into the **pVal** variable created above, using the get_name method of the **Istudent** interface in this example.

```
lrc_Istudent_get_name(Istudent_0, &pVal, CHECK_HRES);
```

3 VuGen then generates a statement for saving the values.

```
//lrc_save_BSTR("param-name",pVal);
```

The statement is commented out. You can remove the comments and change <param-name> to a variable with a meaningful name to be used for storing this value. VuGen will use the variable to save the value of **pVal** returned by the previous call. You can then use the variable as a parameterized input in subsequent calls to other methods.

The IDispatch Interface

Most COM objects have specific interfaces. Many of them also implement a general-purpose interface called **IDispatch**, which VuGen translates in a special way. IDispatch is a "superinterface" that exposes all of the other interfaces and methods of a COM object. Calls to the **IDispatch:Invoke** method from VuGen scripts are implemented using **lrc_Disp** functions. These calls are constructed somewhat differently from calls to other interfaces.

The **IDispatch** interface **Invoke** method can execute a method, it can get a property value, or it can set a value or reference value for a property. In the standard **IDispatch:Invoke** method these different uses are signalled in a **wflags** parameter. In the VuGen implementation they are implemented in different procedure calls that invoke a method or put or get a property.

For example, a call to IDispatch to activate the GetAgentsArray method may look like this:

```
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray", /*locale*/1033,
LAST_ARG, CHECK_HRES);
```

The parameters in the above call are:

IDispatch_0	This is the pointer to the **IDispatch** interface returned by a previous call to the **IUnknown:Queryinterface** method.
GetAgentsArray	This is the name of the method to invoke. Behind the scenes, VuGen will get the ID of the method from the name.
1033	This is the language locale.

LAST_ARG	This is a flag to tell the **IDispatch** interface that there are no more arguments.
CHECK_HRES	This flag turns on checking of HRES, since the call succeeded when it was recorded.

In addition, there might be another parameter, OPTIONAL_ARGS. This signals that in addition to any standard parameters, VuGen is sending some optional arguments. Each optional argument consists of a pair giving the ID or name of the argument and its value. For example, the following call to lrc_DispMethod passes optional arguments "#3" and "var3":

```
{
        GUID riid = IID_IDispatch;
        lrc_IOptional_QueryInterface(IOptional_0, &riid, (void**)&IOptional_0,
CHECK_HRES);
    }
    {
        VARIANT P1 = lrc_variant_short("47");
        VARIANT P2 = lrc_variant_short("37");
        VARIANT P3 = lrc_variant_date("3/19/1901");
        VARIANT var3 = lrc_variant_scode("4");
        lrc_DispMethod((IDispatch*)IOptional_0, "in_out_optional_args", /*locale*/1024,
&P1, &P2, OPTIONAL_ARGS, "#3", &P3, "var3", &var3, LAST_ARG, CHECK_HRES);
}
```

The different **lrc_Disp** methods that use the **IDispatch** interface are detailed in the *Online Function Reference*.

Type Conversions and Data Extraction

As shown in the above example, many COM parameters are defined as variants. To extract these values, VuGen uses a number of conversion functions, derived from the equivalent COM functions. The full list is given in the *Online Function Reference*. Previously, we showed how the **lrc_DispMethod1** call was used to retrieve an array of name strings:

```
VARIANT retValue = lrc_variant_empty();
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray", /*locale*/1033,
LAST_ARG, CHECK_HRES);
```

The following example now shows how VuGen gets the strings out of **retValue**, which is a variant that will be read as an array of strings.

First, VuGen extracts the BSTR array from the variant:

```
BstrArray array0 = 0;
array0 = lrc_GetBstrArrayFromVariant(&retValue);
```

With all the values in array0, VuGen provides you with code that you can use to extract the elements from the array for later use in parameterization, as in the example below:

```
//GetElementFrom1DBstrArray(array0, 0); // value: Alex
//GetElementFrom1DBstrArray(array0, 1); // value: Amanda
....
```

VuGen has numerous type conversion functions and functions for extracting conventional types from variants. These are detailed in the *Online Function Reference* (**Help** > **Function Reference**)

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script and assist you in getting a successful replay. It performs the following steps:

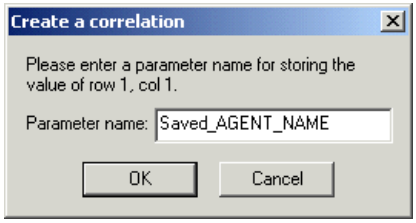
- scans for potential correlations
- insert the appropriate correlation function to save the results to a parameter
- replace the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

To scan and correlate a script detected with automatic correlation:

1. Select **View** > **Output** to display the output tabs at the bottom of the window. Check for errors in the Replay Log tab. Often, these errors can be corrected by correlation.
2. Select **Vuser** > **Scan for Correlations**. VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.
3. Correlate the value. In the Correlated Query tab, double-click on the result you want to correlate. This is located on the line of the message where it says
   grid column x, row x.
   VuGen sends the cursor to the grid location of the value in your script.
4. In the grid, select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



5. Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrc_save**_<type>) which saves the result to a parameter.
6. Click **Yes** to confirm the correlation.
7. A message box opens asking if you want to search for all occurrences of the value in the script.
   Click **No** to replace only the value in the selected statement.
   To search and replace additional occurrences click **Yes**.
8. A Search and Replace dialog box opens. Confirm any replacements, including your original statement.
9. Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. Note that if you cancel the correlation, VuGen also erases the statement created in the previous step.

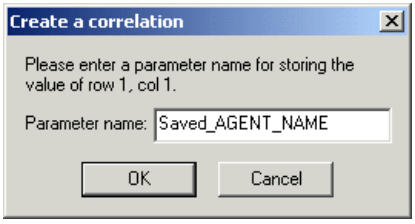
Correlating a Known Value

If you know which value needs to be correlated, perform the following procedure:

To correlate a specific value:

1. Locate the argument you want to correlate (usually in an **lrc_variant_** statement) and select the value without the quotation marks.
2. Select **Vuser** > **Scan for Correlations (at cursor)**.

   VuGen scans the value and lists all results within the script that match this value. The correlation values are listed in the Correlated Query tab.
3. In the Correlated Query tab, double-click on the result you want to correlate. This is located on the line of the message where it says grid column x, row x.
   VuGen sends the cursor to the grid location of the value in your script.
4. In the grid, select the value you want to correlate and select **Vuser** > **Create Correlation**. VuGen prompts you to enter a parameter name for the result value.



5. Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (lrc_save_<type>) which saves the result to a parameter.

```
lrc_save_rs_param (Recordset20_0, 1, 1, 0, "Saved_AGENT_NAME");
```

6. Click **Yes** to confirm the correlation.

7 A message box opens asking if you want to search for all occurrences of the value in the script.
Click **No** to replace only the value in the selected statement.
To search and replace additional occurrences click **Yes**.

8 A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

41

AJAX (Click and Script) Protocol

VuGen allows you to create scripts that emulate AJAX (Asynchronous JavaScript and XML) enabled applications.

This chapter includes:

About Developing AJAX (Click and Script) Vuser Scripts

AJAX (Asynchronous JavaScript and XML) is a technique for creating interactive Web applications. With AJAX, Web pages exchange small packets of data with the server, instead of reloading an entire page. This reduces the amount of time that a user needs to wait when requesting data. It also increases the interactive capabilities and enhances the usability.

Using AJAX, developers can create fast Web pages using Javascript and asynchronous server requests. The requests can originate from user actions, timer events, or other predefined triggers.

AJAX components, also known as AJAX controls, are GUI based controls that use the AJAX technique—they send a request to the server when a trigger occurs.

For example, a popular AJAX control is a **Reorder List** control that lets you drag components to a desired position in a list. VuGen's support for AJAX implementation is based on Microsoft's ASP.NET AJAX Control Toolkit formerly known as Atlas.

The supported frameworks for AJAX functions are:

- Atlas 1.0.10920.0/ASP.NET AJAX—All controls
- Scriptaculous 1.8—Autocomplete, Reorder List, and Slider

VuGen supports the following frameworks at the engine level. This implies that VuGen will create standard Web Click and Script steps, but not AJAX specific functions:

- Prototype 1.6
- Google Web Toolkit (GWT) 1.4

Recording an AJAX (Click and Script) Session

To create a Vuser script that emulates AJAX enabled applications, you select the **AJAX (Click and Script)** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions on the Web page, that are affected by AJAX such a reordering a list. For general information about creating and recording a script, see Chapter 5, "Recording with VuGen."

You can set event related recording options. See Chapter 74, "Click and Script Recording" for more information.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding AJAX (Click and Script) Scripts

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported AJAX controls, VuGen generates a function with an **ajax_xxx** prefix.

In the following example, a user selected item number 1 (index=1) in an Accordion control. VuGen generated an **ajax_accordion** function.

```
web_browser("Accordion.aspx",
                    DESCRIPTION,
                    ACTION,
                    "Navigate=http://labm1app08/AJAX/Accordion/Accordion.aspx",
                    LAST);

lr_think_time(5);

ajax_accordion("Accordion",
                     DESCRIPTION,
                     "Framework=atlas",
                     "ID=ctl00_SampleContent_MyAccordion",
                     ACTION,
                     "UserAction=SelectIndex",
                     "Index=1",
                     LAST);

web_edit_field("free_text_2",
          "Snapshot=t18.inf",
          DESCRIPTION,
          "Type=text",
          "Name=free_text",
          ACTION,
          "SetValue=FILE_PATH",
          LAST);
```

Note: When you record an AJAX session, VuGen generates standard Web (Click and Script) functions for objects that are not one of the supported AJAX controls. In the example above, the word FILE_PATH was typed into an edit box.

42

AMF Protocol

VuGen allows you to create Vusers that emulate Flash Remoting using the AMF format.

This chapter includes:

About Developing AMF Vuser Scripts

Many client applications communicate with servers using RPC (Remote Procedure Calls). RPC, however, presents compatibility and security problems when working over the Internet. Firewalls and proxy servers often block this type of traffic.

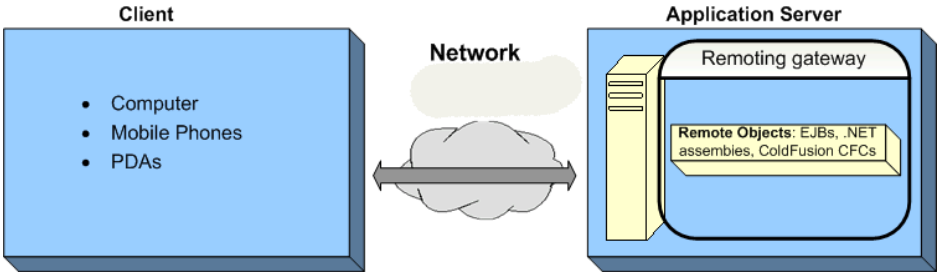
HTTP is supported by all Internet browsers and servers. Therefore, HTTP is a preferred method of communication between client applications and servers when working over the Internet.

SOAP, an XML-based format, provides a secure way to communicate between applications over HTTP. However, since the messages are text-based, SOAP is inefficient when working with large messages such as Flash files and other RIAs (Rich Internet Applications).

To overcome this inefficiency, Macromedia created a proprietary protocol, AMF (Action Messaging Format), which communicates over HTTP in binary format. The binary AMF data set is considerably smaller than that of SOAP's text-based XML.

A typical client application that submits AMF messages to a server is the Flash Player that plays Flash clips on personal computers. The Flash Player sends native Flash objects to an application server via a gateway. The gateway, known as the **Flash Remoting** gateway, is a server-side object, installed on either a Java (including ColdFusion) or .NET server. The gateway acts as a broker that handles requests between the Flash Player and the server. It translates Flash objects into native objects for the server and passes them on to the appropriate server-side services.

After the results are returned, the gateway serializes them back into native Flash objects and sends them to the Flash client via AMF.



To create a Vuser script that emulates the AMF traffic, you select the **AMF** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions against the Web server. For general information about creating and recording a script, see Chapter 5, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding AMF Terms

The following table provides definitions for the most common terms that relate to AMF:

Term/ Abbreviation	Description
ActionScript	A script programming language used for controlling Flash movies and applications. Its syntax is similar to JavaScript.
AMF	A proprietary binary communication protocol used for Flash Remoting.
Flash Remoting	Flash Remoting allows data to be exchanged between a Flash Player and an application server using the AMF format.
Flex	An application server for generating RIAs (Reach Internet Applications).
SOAP	A standard for exchanging XML-based messages over a computer network, normally using HTTP.

Working with AMF Functions

When you record a Flash Remoting session, VuGen generates AMF Vuser script functions that emulate your actions. All AMF functions begins with an **amf** prefix.

In the following example, the **amf_define_header_set** function defines a header set. The **amf_call** function accesses a gateway and sends a message to the server.

```
amf_define_header_set("Id=amf_header_set",
    HEADER,
    "Name=amf_server_debug",
    "MustUnderstand=true",
    "Data=<object><boolean key=\"coldfusion\">true</boolean>
            <boolean key=\"""amfheaders\">false</boolean>…
    LAST);

amf_call("flashgateway.samples.FlashJavaBean.testDocument",
    "Gateway=http://testlab:8200/flashservices/gateway",
    "AMFHeaderSetId=amf_header_set",
    "Snapshot=t3.inf",
    MESSAGE,
    "Method=flashgateway.samples.FlashJavaBean.testDocument",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    "<xmlString><![CDATA[<TEST message=\"test\"><INSIDETEST
      /></TEST>]]></"
    "xmlString>",
    END_ARGUMENTS,
    LAST);
```

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Correlating AMF Scripts

Flash Applications often contain dynamic data, data that changes each time you run the script. For example, certain servers use links comprised of the current date and time. Alternatively, the object name may change from run to run.

When you record a Vuser script, VuGen records a set of data and argument values. When you replay the script, however, the server may reject these arguments and issue an error. This error could be the result of dynamic data that is outdated and no longer accepted by the server.

To overcome this, you apply correlation to your script:

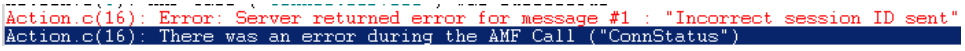
- Save the server response in preparation for extracting the required values.
- Extract the required values from the server response.
- Save the values to a parameter.
- Use those parameters as input to your AMF requests.

These errors are not always obvious, and you may only detect them by carefully examining Vuser log files. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

To perform correlation:

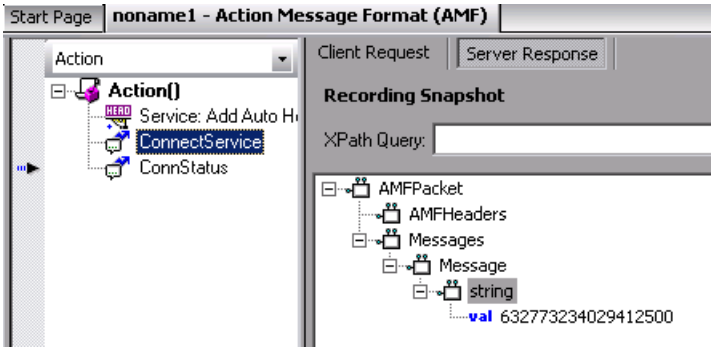
1 Locate the step in your script that failed due to dynamic values that need correlation.

Use the Replay Log to assist you in finding the problematic step.



2 Identify the server response with the correct value in one of the previous steps.

Examine the proceeding steps in Tree View and look for the value in the **Server Response** tab.



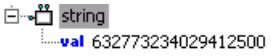
3 Save the entire server response to a parameter.

Before you extract the value, the entire server response should be saved to a parameter as follows:

- Right-click the step node (in the Action pane) corresponding to the server response containing the value and select **Properties**.
- In the AMF Call Properties dialog, type a **Response parameter** name. For details, see "AMF Call Properties" on page 706.
- Click **OK** to save the new parameter name.

4 Save the original server response value to a parameter.

- In the XML tree of the Server Response, right-click the node above the value (for example, string), and select **Save value in parameter**.

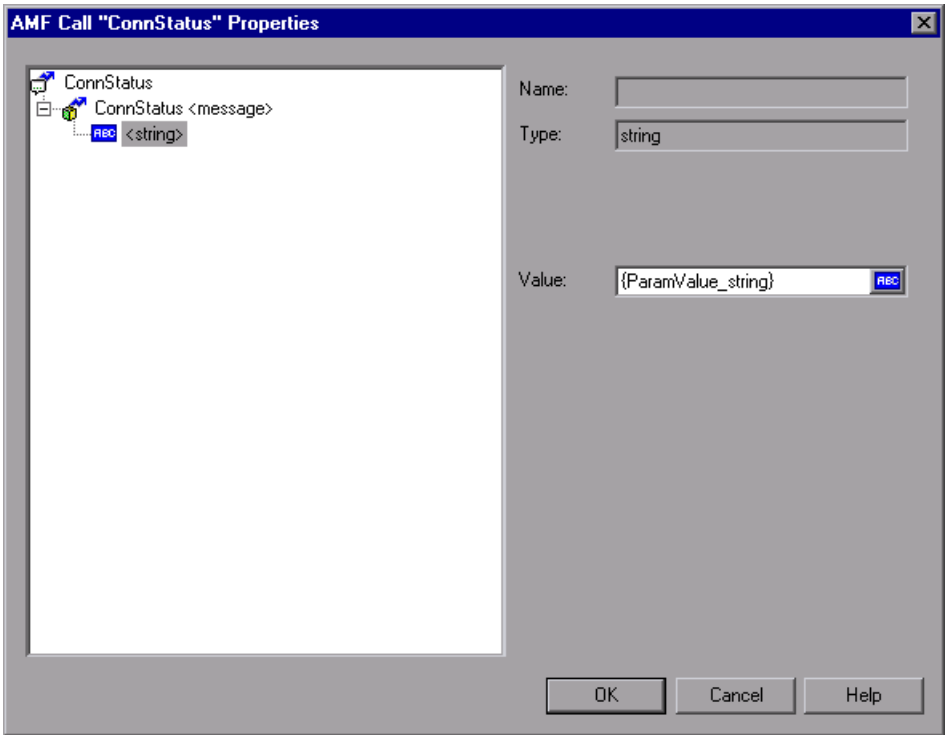


- In the XML Parameter Properties dialog, specify a parameter **Name**. You will use this name in subsequent steps.
- Click **OK**. The script will now contain a new function, **lr_xml_get_values**.

5 Insert the parameter in the subsequent calls.

In VuGen edit view, beginning with the call that failed, replace the value in all subsequent calls to the object with the parameter that you defined:

- Right-click the step node (in the Action pane) corresponding to the failed call and select **Properties**.
- Locate the argument that required correlation.
- In the **Value** box, type the parameter name in curly brackets, for example, **{ParamValue_string}**.



- Click **OK**.

6 Run the script.

Make sure that VuGen properly substitutes the argument value with the parameter value that you saved.

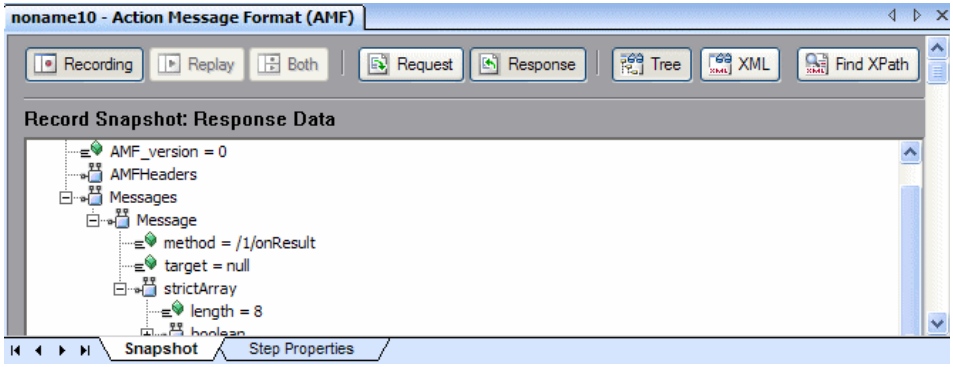
In the following example, **lr_xml_get_values** retrieves the value from the response and creates the parameter ParamValue_string. This parameter, ParamValue_string, was used in the next **amf_call** function.

```
amf_call(
    "ConnectService",
    "Gateway=http://labm1app08/AMF/EchoAMF/gateway.aspx",
    "Snapshot=t6.inf",
    "ResponseParameter=resp",
    MESSAGE,
    "Method=EchoAMF.SpecialCases.ConnectService",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    END_ARGUMENTS,

    LAST);

lr_xml_get_values("XML={resp}",
    "FastQuery=/AMFPacket/Messages/Message/string",
    "ValueParam=ParamValue_string",
    LAST);

amf_call(
    "ConnStatus",
    "Gateway=http://labm1app08/AMF/EchoAMF/gateway.aspx",
    "Snapshot=t7.inf",
    MESSAGE,
    "Method=EchoAMF.SpecialCases.ConnStatus",
    "TargetObjectId=/2",
    BEGIN_ARGUMENTS,
    "<string>{ParamValue_string}</string>",
    END_ARGUMENTS,

    LAST);
```

For more information about correlation for Web Vusers, see Chapter 53, "Web (HTTP/HTML) Correlation Rules."

Viewing AMF Data

To view the AMF data in XML format, select the appropriate step in the Tree View. The right pane of VuGen displays the client request and server response in XML hierarchy. To view it in true XML, click the **XML** button.

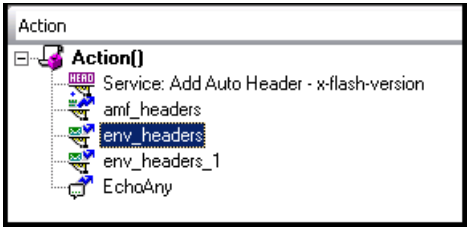


To view all the elements of the message, make sure that the nodes of the XML are expanded. To view the XML argument values in a grid, select **Show node values in grid**.

VuGen provides a utility to find XML through its XPath. You can also use the Query Builder to assist you in creating the queries. For more information, see "Querying an XML Tree" on page 719.

Understanding AMF Scripts

In the following example, the AMF script contains an AMF call, AMF header, and AMF envelope header. To view details of each node, you select a node and select Properties from the right-click menu.

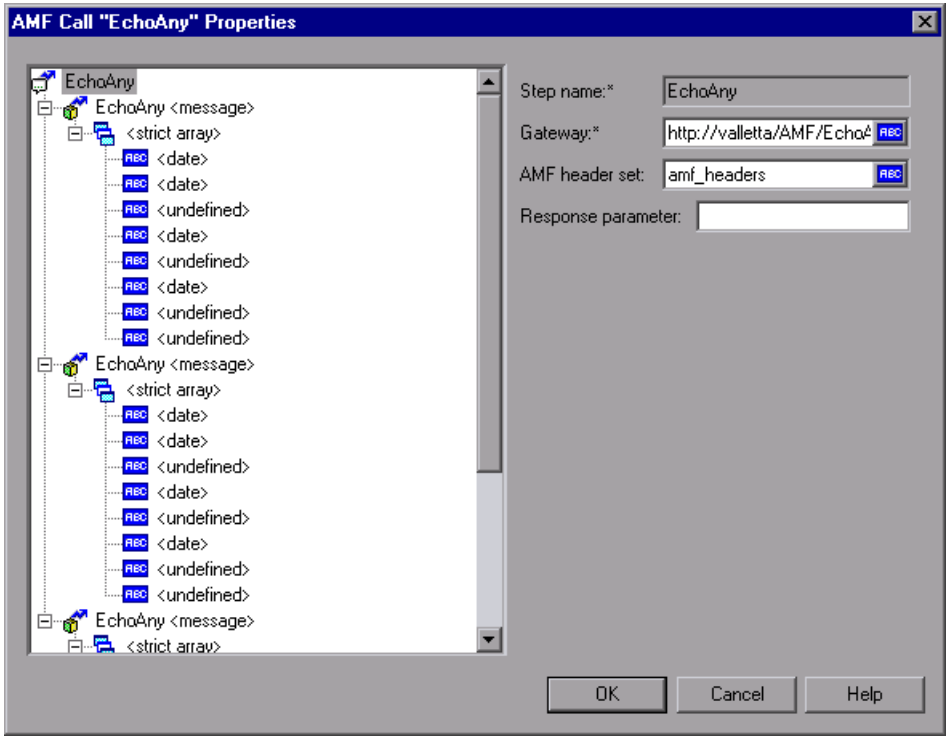


The following sections describe the AMF node properties:

- AMF Call Properties
- AMF Header Set Properties
- AMF Envelope Header Set Properties

AMF Call Properties

The AMF Call Properties dialog shows the AMF request details: the AMF call, one or more messages, and the arguments for each message. In the example below, the AMF call is **EchoAny**, and it is associated with the **amf_headers** header set.

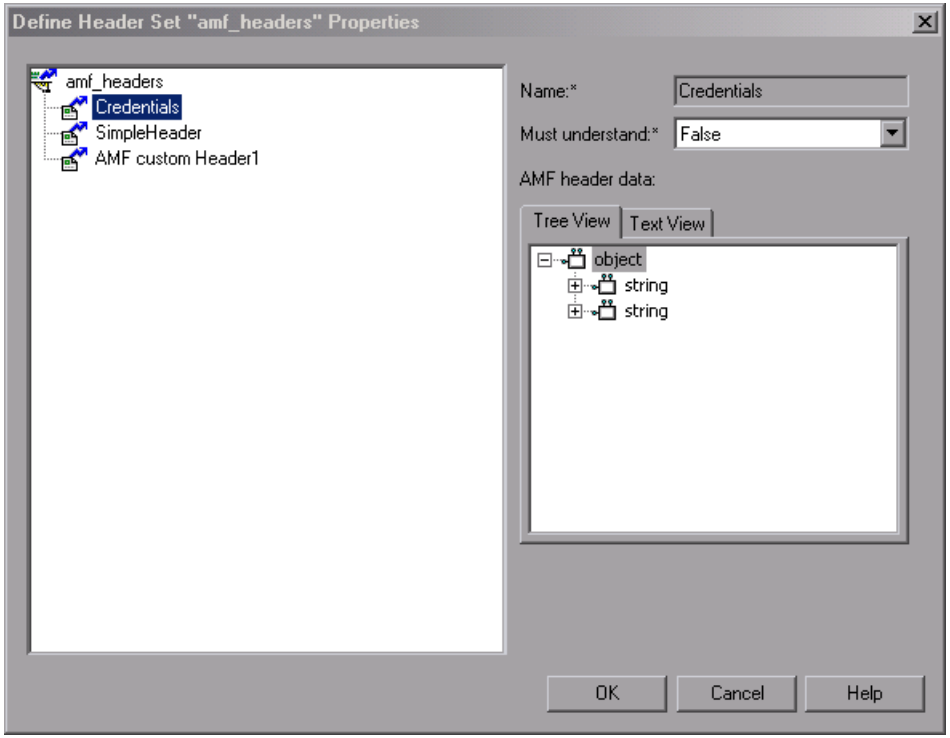


Each AMF call has three levels:

- The AMF Call level contains a step name, gateway, AMF header set, and response parameter
- The AMF Message level contains a method, target object ID, and Envelope header set
- The AMF Argument level contains a name, type, and value

AMF Header Set Properties

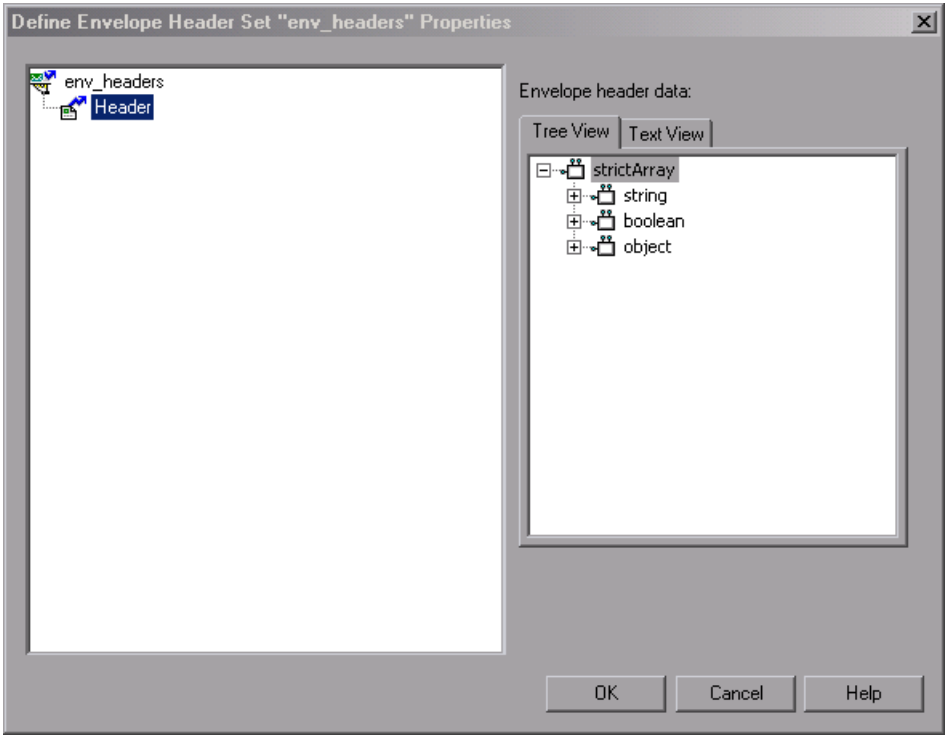
The AMF Header Set Properties dialog shows the AMF header set definition: One or more AMF header names and associated header data in XML format. In the example below, the header set has three headers.



Each header has a **Must understand** argument which indicates whether processing the call continues if the server cannot interpret the header. If MustUnderstand is true, this header is mandatory and processing is aborted if it is not understood.

AMF Envelope Header Set Properties

The AMF Envelope Header Set Properties dialog shows the AMF envelope header set and corresponding data for each header.



43

FTP Protocol

VuGen allows you to emulate network activity by directly accessing an FTP (File Transfer Protocol) server.

This chapter includes:

About Developing FTP Vuser Scripts

The FTP protocol is a low-level protocol that allows you to emulate the actions of a user working against an FTP server.

For FTP, you emulate users logging into an FTP server, transferring files, and logging out. To create a script, you can record an FTP session or manually enter FTP functions.

When you record an FTP session, VuGen generates functions that emulate the mail client's actions. If the communication is performed through multiple protocols such as FTP, HTTP, and a mail protocol, you can record all of them. For instructions on specifying multiple protocols, see Chapter 5, "Recording with VuGen."

To create a script for the FTP protocol, you select the FTP protocol type in the E-Business category. To begin recording, you click the **Record** button and perform typical actions against the FTP server. For more information on creating and recording a script, see Chapter 5, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availability Center* documentation.

Working with FTP Functions

When recording a FTP session, VuGen generates FTP functions. Each FTP function begins with an **ftp** prefix.

Most FTP functions come in pairs—one for global sessions and one where you can indicate a specific mail session. To apply the action to all sessions, use the version without the ex suffix. To apply the action to a specific session, use the version with the session identifier with the ex suffix. For example, **ftp_logon** logs on to the FTP server globally, while **ftp_logon_ex** logs on to the FTP server for a specific session.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 76, "Setting Script Generation Preferences."

In the following example, the ftp_delete function deletes the test.txt file from the FTP server.

```
ftp_logon("FTP",
                    "URL=ftp://user:pwd@ftp.merc-int.com",
                    "LocalAddr=ca_server:21",
                    LAST);

ftp_delete("Ftp_Delete",
        "PATH=/pub/for_jon/test.txt", ENDITEM,
        LAST);

ftp_logout();
```

44

Flex Protocol

VuGen allows you to create Vusers that emulate the protocol suite provided with the Flex 2 SDK.

This chapter includes:

About Developing Flex Vuser Scripts

Flex is a collection of technologies that provide developers with a framework for building RIAs (Rich Internet Applications) based on the Flash Player.

RIAs are lightweight online programs that provide users with more dynamic control than with a standard web page. Like Web applications built with AJAX, Flex applications are more responsive, because the application does not need to load a new Web page every time the user takes action. However, unlike working with AJAX, Flex is independent of browser implementations such as JavaScript or CSS. The framework runs on Adobe's cross-platform Flash Player.

Flex 2 applications consist of many MXML and ActionScript files. They are compiled into a single SWF movie file which can be played by Flash player, installed on the client's browser.

Flex 2 supports a variety of client/server communication methods, such as RPC, Data Management, and Real-Time messaging. It supports several data formats such as HTTP, AMF, and SOAP.

VuGen lets you create a Vuser script that emulates communication with Flex 2 RPC services. VuGen's **Flex** type lets you create scripts that emulate Flex applications working with AMF3 or HTTP data. For Flex applications working with SOAP data, use the **Web Services** Vuser type.

The following sections describe how to create scripts for applications using AMF3 or HTTP communication. For information about Web Service Vuser scripts, see Chapter 13, "Web Services Protocol."

To create a script, you select the **Flex** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions in your Flex application. For general information about creating and recording a script, see Chapter 5, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Working with Flex Functions

When you record a Flex application, VuGen generates Flex Vuser script functions that emulate your application. The following functions represent some of the Flex remoting steps:

Function Name	Description
flex_login	Logs on to a password-protected Flex application.
flex_logout	Logs off of a password-protected Flex application.
flex_ping	Checks if a Flex application is available.
flex_remoting_call	Invokes one or more methods of a server-side Remote object (RPC).
flex_web_request	Sends an HTTP request with any method supported by HTTP.
flex_amf_call	Sends an AMF request.
flex_amf_define_header_set	Defines a set of AMF headers.
flex_amf_define_envlope_header_set	Defines a set of envelope headers.

In the following example, **flex_ping** checks for the availability of a service. The **flex_remoting_call** function invokes the service remotely.

```
flex_ping("1",
    "URL=http://testlab1/weborb30/console/weborb.aspx",
    "Snapshot=t6.inf",
    LAST);

flex_remoting_call("getProductEdition::GenericDestination",
    "URL=http://testlab1/weborb30/console/weborb.aspx",
    "Snapshot=t7.inf",
    INVOCATION,
    "Target=/2",
    "Operation=getProductEdition",
    "Destination=GenericDestination",
    "DSEndpoint=my-amf",
    "Source=Weborb.Management.LicenseService",
    "Argument=<arguments/>",
    LAST);
```

For detailed syntax information about all of the Flex functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Correlating Flex Scripts

Flex applications often contain dynamic data, data that changes each time you run the script. For example, the object name may change from run to run.

When you record a Vuser script, VuGen records a set of data and argument values. When you replay the script, however, the server may reject these arguments and issue an error. This error could be the result of dynamic data that is outdated and no longer accepted by the server.

To overcome this, you apply correlation to your script:

- Save the server response in preparation for extracting the required values.
- Extract the required values from the server response.
- Save the values to a parameter.
- Use those parameters as input to your Flex requests.

These errors are not always obvious, and you may only detect them by carefully examining Vuser log files. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

To perform correlation:

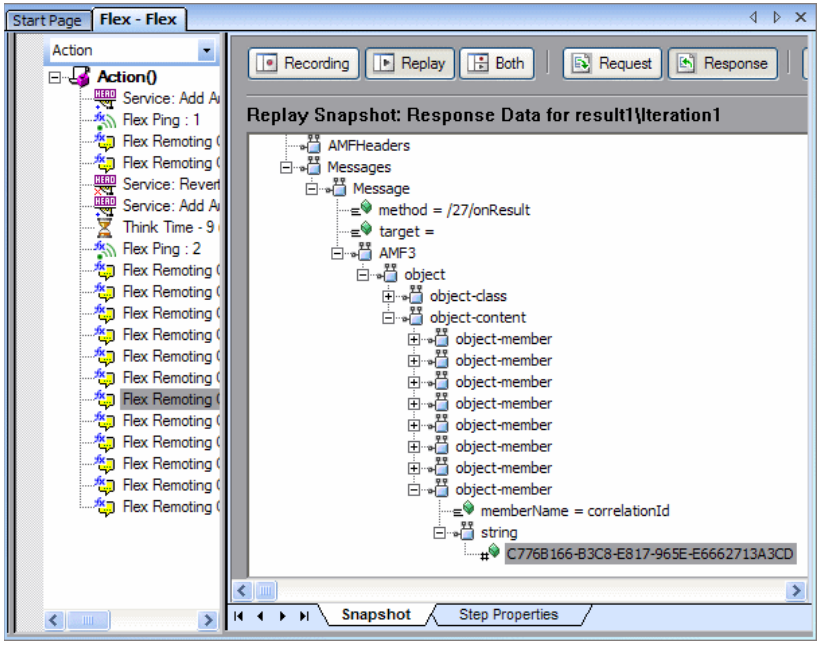
1 Locate the step in your script that failed due to dynamic values that need correlation.

Use the Replay Log to assist you in finding the problematic step.

```
Action.c(16): Error Server returned error for message #1 : "Incorrect session ID sent"/
Action.c(16): There was an error during the Flex Call ("ConnStatus")
```

2 Identify the server response with the correct value in one of the previous steps.

Double-Click the error in the Replay log to go to the step with the error. Examine the preceding steps in Tree View and look for the value in the **Server Response** tab.



3 Save the entire server response to a parameter.

Before you extract the value, the entire server response should be saved to a parameter as follows:

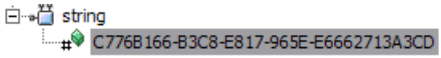
➤ Right-click the step node (in the left Action pane) corresponding to the server response containing the value and select **Properties**.

➤ In the Flex Call Properties dialog box, type a **Response parameter** name.

➤ Click **OK** to save the new parameter name.

4 Save the original server response value to a parameter.

➤ In the **Replay Snapshot: Response Data**, right-click the node above the value (for example, string), and select **Save value in parameter**.



➤ In the XML Parameter Properties dialog, specify a parameter **Name**. You will use this name in subsequent steps.

➤ Click **OK**. The script will now contain a new function, **lr_xml_get_values**.

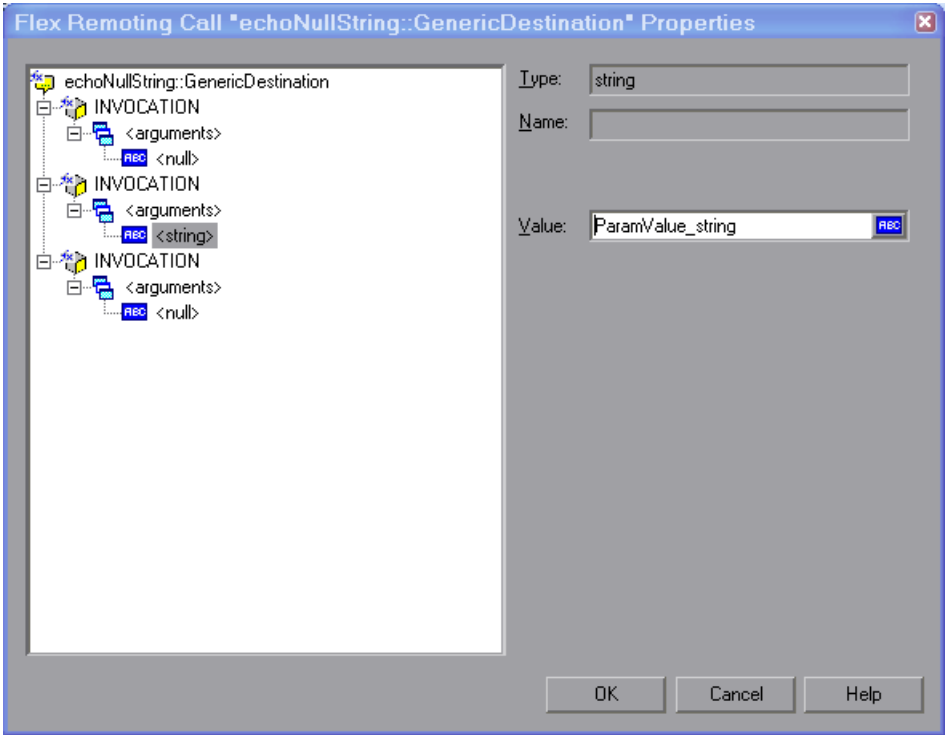
5 Insert the parameter in the subsequent calls.

In VuGen edit view, beginning with the call that failed, replace the value in all subsequent calls to the object with the parameter that you defined:

➤ Right-click the step node (in the Action pane) corresponding to the failed call and select **Properties**.

➤ Locate the argument that required correlation.

➤ In the **Value** box, type the parameter name in curly brackets, for example, **{ParamValue_string}**.



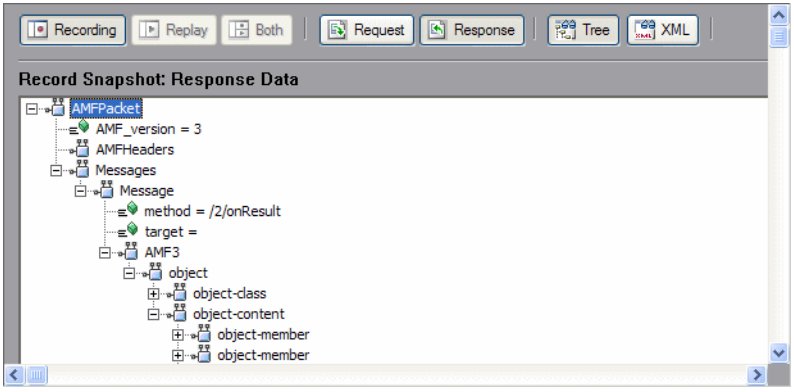
Click **OK**.

6 Run the script.

Make sure that VuGen properly substitutes the argument value with the parameter value that you saved.

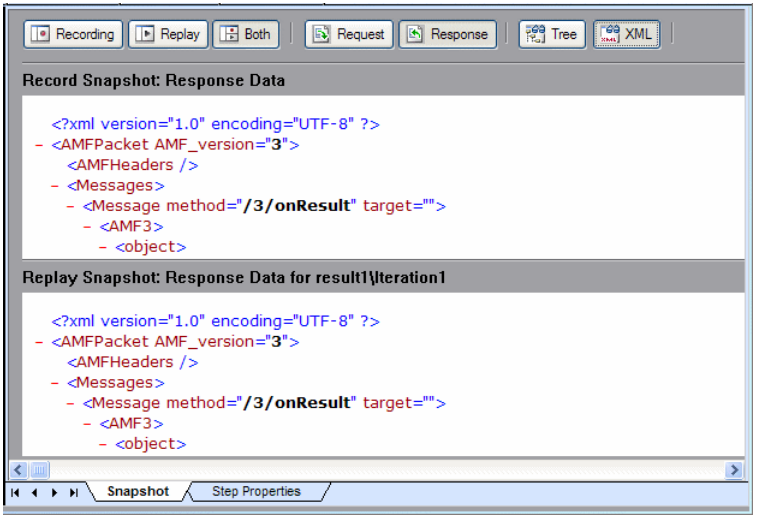
Viewing Flex Data

To view the Flex data, select the appropriate step in the Tree View. The right pane of VuGen displays a snapshot of the data.



To view all the elements of the message, expand the desired nodes. You can view the Request or Response data for Recording and Replay, by clicking the appropriate buttons in the snapshot.

To view the data in its XML structure, click the **XML** button in the snapshot.



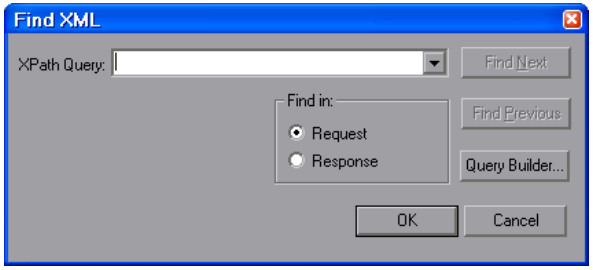
Querying an XML Tree

VuGen provides a Query Builder that lets you create and execute queries on the XML.

VuGen displays the XML code in an expandable tree. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

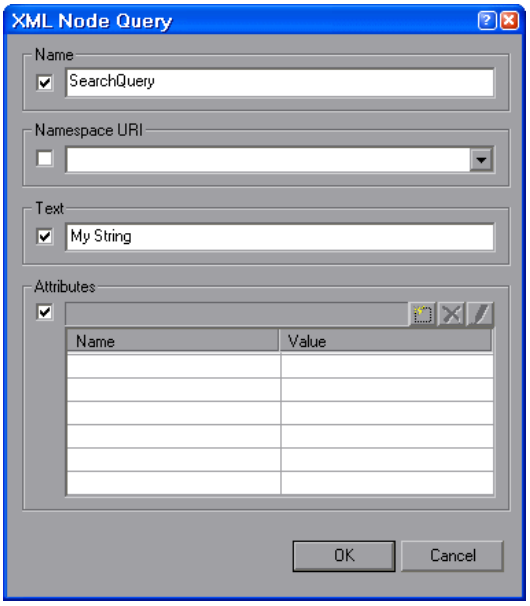
To perform a query:

1 In the Snapshot tab, select on the node that you want to search. Click the **Find XML** button. The Find XML dialog button opens.



2 Select Request or Response. Enter an XPath query and click **OK**. To formulate a query, click **Query Builder** button. The XML Node Query dialog box opens.

3 Enable one or more items for searching.



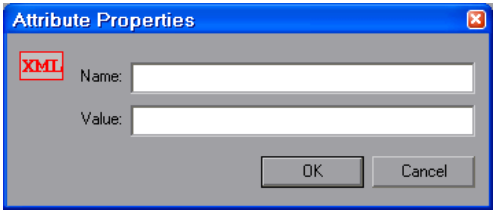
4 Enable the **Name** section to search for the name of a node or element.

5 Enable the **Namespace URI** section to search for a namespace.

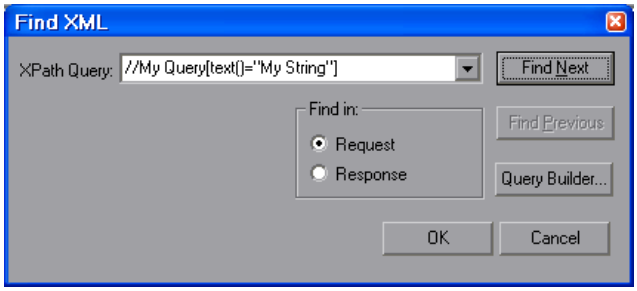
6 Enable the **Text** section to search for the value of the element indicated in the Name box.

7 Enable the **Attributes** section to search for an attribute.

8 Enter the search text in the appropriate boxes. To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.



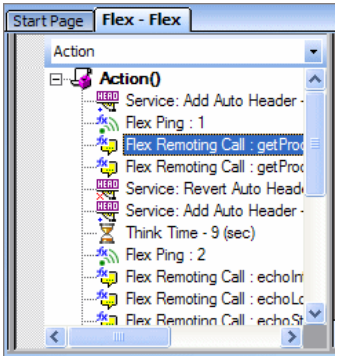
9 Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the Find XML box.



10 Click **Find Next** to begin the search.

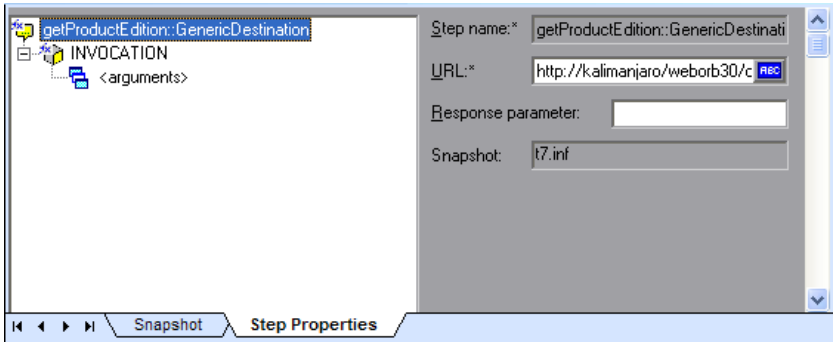
Setting Flex Step Properties

The Flex Vuser script contains a series of Flex calls. You can view information about each step by selecting it in the Tree view.



The right pane has two tabs (for most Flex steps)—Snapshot and Properties The Snapshot tab shows the data, while the Properties step displays the properties each step:

The following example shows the properties for a **Flex Remoting Call** step.



As you select each node, the right pane shows the properties specific for that node.

Working with Flex RTMP

Flex can record and replay scripts involving RTMP (Real Time Messaging Protocol). In order to enable RTMP simulation, you must configure the recording options for the Flex protocol.

To enable RTMP:

1. Open the Recording Options dialog box by selecting **Tools** > **Recording Options** or clicking the **Options** button in the Start Recording dialog box.
2. In the **Network** > **Port Mapping** node click **Options**.
3. Set the **Send-Receive buffer size threshold** to 1500.

45

LDAP Protocol

VuGen allows you to emulate the communication with an LDAP server.

This chapter includes:

About Developing LDAP Vuser Scripts

LDAP, the Lightweight Directory Access Protocol, is a protocol used to access a directory listing. The LDAP directory is composed of many LDAP entries. Each LDAP entry is a collection of attributes with a name, called a distinguished name (DN). For more information about DN, see "Defining Distinguished Name Entries" on page 726.

LDAP directory entries are arranged in a hierarchical structure that reflects political, geographic, and/or organizational boundaries. Entries representing countries appear at the top of the tree. Below them are entries representing states or national organizations. Below them might be entries representing people, organizational units, printers, documents, or just about anything else.

VuGen records communication over LDAP servers. It creates a script, with functions that emulate your actions. This includes logging in and out of the server, adding and deleting entries, and querying an entry.

To create a script for the LDAP protocol, you select the LDAP protocol type in the E-Business category. To begin recording, select **Vuser > Start Recording**, and perform typical actions against the LDAP server. For more information on the recording procedure, see Chapter 5, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availability Center* documentation.

Working with LDAP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 76, "Setting Script Generation Preferences."

LDAP Vuser script functions emulate the LDAP protocol. Each LDAP function begins with the **mldap** prefix.

All LDAP functions come in pairs—one for global sessions and one where you can indicate a specific session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the ex suffix. For example, **mldap_logon** logs on to the LDAP server globally, while **mldap_logon_ex** logs on to the LDAP server for a specific session.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the user logs on to an LDAP server, ldap1. It adds an entry and then renames the OU attribute from Sales to Marketing.

```
Action()
{
                // Logon to the LDAP server
                mldap_logon("Login",
                                "URL=ldap://johnsmith:tiger@ldap1:80",
                                LAST);

                // Add an entry for Sally R. Jones
                mldap_add("LDAP Add",
                          "DN=cn=Sally R. Jones,OU=Sales, DC=com",
                          "Name=givenName", "Value=Sally", ENDITEM,
                          "Name=initials", "Value=R", ENDITEM,
                          "Name=sn", "Value=Jones", ENDITEM,
                          "Name=objectClass", "Value=contact", ENDITEM,
                          LAST);

                // Rename Sally's OU to Marketing
                mldap_rename("LDAP Rename",
                             "DN=CN=Sally R. Jones,OU=Sales,DC=com",
                             "NewDN=OU=Marketing",
                             LAST);

                // Logout from the LDAP server
                mldap_logoff();
    return 0;
}
```

Defining Distinguished Name Entries

The LDAP API references objects by its **distinguished name** (DN). A DN is a sequence of **relative distinguished names** (RDN) separated by commas.

An RDN is an attribute with an associated value in the form attribute=value. The attribute names are not case-sensitive. The following table lists the most common RDN attribute types.

String	Attribute Type
DC	domainComponent
CN	commonName
OU	organizationalUnitName
O	organizationName
STREET	streetAddress
L	localityName
ST	stateOrProvinceName
C	countryName
UID	userid

The following are examples of distinguished names:

```
DN=CN=John Smith,OU=Accounting,DC=Fabrikam,DC=COM
DN=CN=Tracy White,CN=admin,DC=corp,DC=Fabrikam,DC=COM
```

The following table lists reserved characters that cannot be used in an attribute value.

Character	Description
	space or # character at the beginning of a string
	space character at the end of a string
,	comma

+	plus sign
"	double quote
\	backslash
<	left angle bracket
>	right angle bracket
;	semicolon

To use a reserved character as part of an attribute value, you must precede it with an escape character, a backslash (\). If an attribute value contains other reserved characters, such as the equal sign (=) or non-UTF-8 characters, you must encode it in hexadecimal format—a backslash followed by two hex digits.

The following are examples of DNs that include escaped characters. The first example is an organizational unit name with an embedded comma; the second example is a value containing a carriage return.

```
DN=CN=Bitwise,OU=Docs\, Support,DC=Fabrikam,DC=COM
DN=CN=Before\0DAfter,OU=Test,DC=North America,DC=Fabrikam,DC=COM
```

Specifying Connection Options

Using the **mldap_logon[_ex]** function, you control the way you login to the LDAP server.

When specifying the URL of the LDAP server, you specify how to connect and with what credentilals.

When specifying the server's URL, use the following format:

```
ldap[s][username:[password]@][server[:port]]
```

The following table shows several examples of connections to LDAP servers.

Syntax	Description
ldap://a:b@server.com:389	Connects to the server (to 389 port) and then binds with username "a" , password "b"
ldap://:@server.com	Connects to server (to default unsecured port 389) then binds anonymously with a NULL username and password
ldaps://a:@server.com	Connects to server (to default secured port 636)and then binds with username "a", password ""
ldap://@server.com, ldap://server.com	Connects to server without binding
ldap://a:b@	Binds with username "a", password "b, executing a bind on the existing session without reconnecting
ldap://:@	Binds anonymously with a NULL username and password (executes bind on existing session without reconnecting)

You can also specify LDAP modes or SSL certificates using the following optional arguments:

- **Mode.** The LDAP call mode: *Sync* or *Async*
- **Timeout.** The maximum time in seconds to search for the LDAP server.
- **Version.** The version of the LDAP protocol version 1,2, or 3
- **SSLCertDir.** The path to the SSL certificates database file (cert8.db)
- **SSLKeysDir.** The path to the SSL keys database file (key3.db)
- **SSLKeyNickname.** The SSL key nickname in the keys database file
- **SSLKeyCertNickname.** The SSL key's certificate nickname in the certificates database file
- **SSLSecModule.** The path to the SSL security module file (secmod.db)
- **StartTLS.** Requires that the StartTLS extension's specific command must be issued in order to switch the connection to TLS (SSL) mode

For detailed information about these arguments, see the *Online Function Reference* (**Help** > **Function Reference**).

46

Microsoft .NET Protocol

VuGen records applications that were created in the .NET Framework environment.

This chapter includes:

Before recording a script, we recommend that you read Chapter 47, "Microsoft .NET Filters." These guidelines will help you create an optimal script that accurately emulates your application.

About Recording Microsoft .NET Vuser Scripts

Microsoft .NET Framework provides a solid foundation for developers to build various types of applications such as ASP.NET, Windows Forms, Web Services, distributed applications, or applications that combine several of these models.

VuGen supports .NET as an application level protocol. It allows you to create Vuser scripts that emulate users of Microsoft .NET client applications created in its .NET Framework. VuGen records all of the client actions through methods and classes, and creates a script in C Sharp or VB .NET.

By default, the VuGen environment is configured for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation) applications. Contact Customer Support for information on how to configure VuGen to record applications created with other client-server activity.

For more information about .NET and the above environments, see the MSDN Web site, http://msdn2.microsoft.com.

Limitations

The following limitations apply to the VuGen recording of a Microsoft .NET application:

- Microsoft .NET scripts only support single-protocol recording in VuGen.
- Direct access to public fields is not supported—the AUT must access fields through methods or properties.
- VuGen does not record static fields in the applications—it only records methods within classes.
- Multi-threaded support is dependent on the client application. If the recorded application supports multi-threading, then the Vuser script will also support multi-threading.
- In certain cases, you may be unable to run multiple iterations without modifying the script. Objects that are already initialized from a previous iteration, cannot be reinitialized. Therefore, to run multiple iterations, make sure to close all of the open connections or remoting channels at the end of each iteration.

- Recording is not supported for Enterprise Services communication based on MSMQ and Enterprise Services hosted in IIS.
- VuGen partially supports the recording of WCF services hosted by the client application.
- Recording is not supported for Remoting calls using a custom proxy.
- Recording is not supported for **ExtendedProperties** property of ADO.NET objects, when using the default ADO.NET filter.
- Applications created with .NET Framework 1.1 which are not compatible with Framework 2.0, cannot be recorded. To check if your Framework 1.1 application is compatible, add the following XML tags to your application's .config file:

```
<configuration>
    <startup>
        <supportedRuntime version="v2.0.50727"/>
    </startup>
</configuration>
```

Invoke the application (without VuGen) and test its functionality. If the application works properly, VuGen can record it. Remove the above tags before recording the AUT with VuGen. For more information regarding this solution, see the MSDN Knowledge Base.

Getting Started with Microsoft .NET Vusers

This section describes the process of developing Microsoft .NET Vuser scripts. Note that the AUT (Application Under Test) must be installed on the machine running the script and all load generators.

To develop a basic .NET Vuser script:

1 Record the application using VuGen.

Start VuGen and create a new Vuser script. Specify **Microsoft .NET** as the type of Vuser. Select an application to record and set the recording options. For more information, see "Microsoft .NET Recording Options" on page 1149.

For general details about recording, see Chapter 5, "Recording with VuGen."

2 Debug the script and set the filter.

Examine the recorded script and determine if the required methods were recorded. If necessary, modify the filter and record the script again.

For details about setting the filter, see "Guidelines for Setting Filters" on page 755.

3 Correlate the script.

Correlate the values in your script to save them as parameters for use at a later point in the script.

For details, see "Correlating Microsoft .NET Scripts" on page 736.

4 Run the script from VuGen.

Save and run the script in VuGen to verify that it runs correctly. To insure a successful replay, compile and run the script in VuGen before running it remotely on a Load Generator machine or via the Controller. This applies each time you make a change to the script—either in the script's settings or the actual script. In addition, if you save the script under another name, replay it in VuGen before running it on a Load Generator.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

5 Configure the run-time settings.

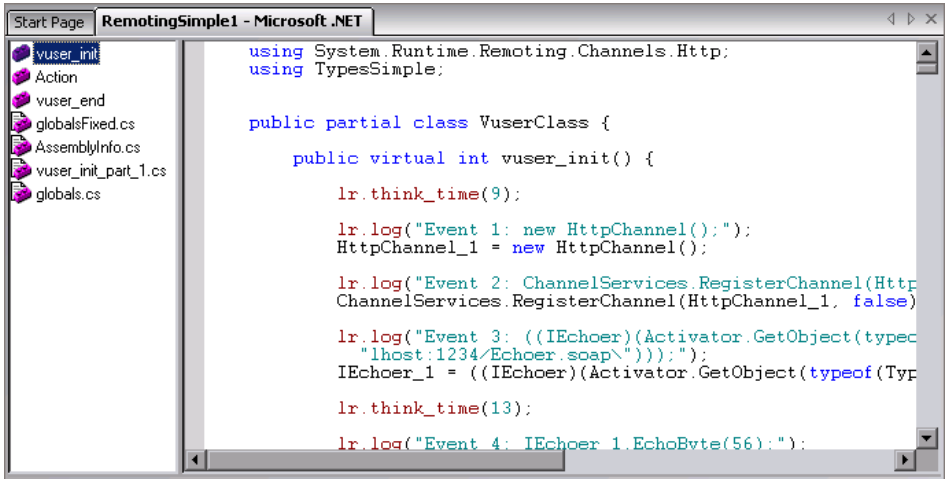
The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see ".NET Environment Run-Time Settings" on page 1315.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Viewing Scripts in VuGen and Visual Studio

After the recording, you can view the script in VuGen's Script view.



VuGen records the actions that occurred and generates C Sharp or VB code. By default, VuGen wraps all of the steps in **lr.log** calls.

When you replay the script, VuGen compiles it first to make sure that all of the calls are valid and that the syntax is correct. VuGen compiles the script into a DLL file, **Script.dll**, saved in the script's **bin** folder. This DLL file contains three functions - Init, Actions, and End.

You can compile the script to check its syntax, without running it. To compile the script directly from VuGen, click Shift+F5 or select **Vuser > Compile**. If VuGen detects a compilation error, it displays it in the Output window. Double-click on the error to go to the problematic line in the script.

To run the script directly from VuGen, click F5 or select **Vuser > Run**. Breakpoints and step-by-step replay are not supported in VuGen's editor window for Microsoft .NET Vusers. To debug a script and run it with breakpoints or step-by-step, run it from within Visual Studio .NET as described below.

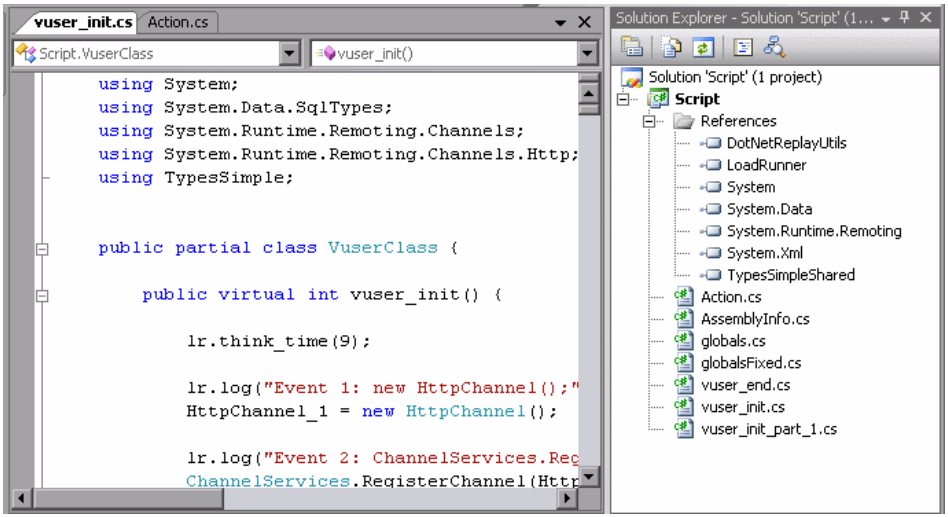
Viewing Scripts in Visual Studio

Visual Studio provides you with additional tools to view, edit, and debug your script. You can add breakpoints, view variable values, add assembly references, and edit the script using Visual Studio's IntelliSense. You can also run the script in a step-by-step mode for debugging.

When you save your script, VuGen creates a Visual Studio 2005 solution file, **Script.sln**, in your script's folder. You can open the solution file in Visual Studio .NET and view all of its components in the Solution Explorer.

To open the solution in Visual Studio 2005, select **Vuser** > **Open Solution in Visual Studio** or click the Visual Studio button on VuGen's toolbar.

Double-click the appropriate section in the Solution Explorer, such as **vuser_init.cs**, to view the contents of the script.



Note that VuGen automatically loads all of the necessary references that were required during recording. You can add additional references for use during compilation and replay through the Solution Explorer. Select the **Reference** node and select **Add Reference** from the right-click menu.

Click on **globals.cs** or **globals.vb** in the Solution Explorer to view a list of the variables defined and used by your script.

Viewing Data Sets and Grids

When you record a method returning a dataset, data table, or data reader action, VuGen generates a grid for displaying the data.

When working with a data reader, VuGen collects the data retrieved from each **Read** operation and converts it to the replay helper function, **DoDataRead**.

For example, after recording the following application code,

```
SqlDataReader reader = command.ExecuteReader();
    while( reader.Read() )
     {
        // read the values, e.g., get the string located in column 1
        string str = reader.GetString(1)
     }
```

VuGen generates the following lines in the script:

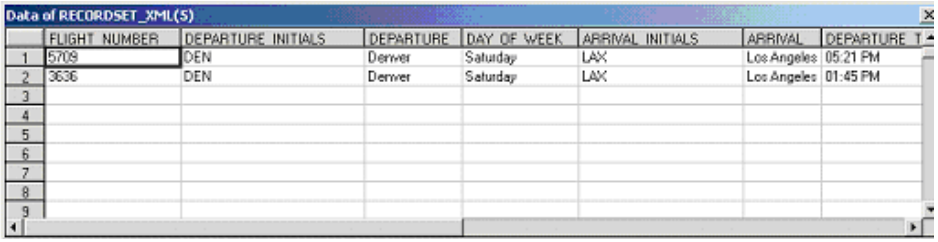
```
SqlDataReader_1 = SqlCommand_1.ExecuteReader();
LrReplayUtils.DoDataRead(SqlDataReader_1, out valueTable_1, true, 27);
```

where two the parameters indicate that during recording, the Application read all 27 available records. Therefore, during replay the script will read all available records.

In addition, VuGen generates a data grid containing all the information retrieved by the **Read** operations.

During replay you can use the output data table, containing the actual retrieved values, for correlation and verification. For more information regarding the **DoDataRead** function, see the *Online Function Reference* (**Help > Function Reference**).

By default, VuGen displays the grids in your script. To disable the grid display and instruct VuGen to show the collapsed version of the grid, select **View** > **Data Grids**.



The dataset is stored in an XML file. You can view this XML file in the script's data/datasets folder. The data files are represented by an <*index_name*>.xml file, such as 20.xml. Since one file may contain several data tables, see the file **datasets.grd** file, which maps the script index to the file index to determine which XML contains the data.

For additional information about grids, see "Working with Grids" on page 586.

Correlating Microsoft .NET Scripts

After you record a session, you may need to **correlate** one or more values within your script. Correlating a value means that you capture a value during the script replay, and save it to a parameter. You can then use this parameter at a later point in the script.

VuGen automatically does a basic correlation—if an object is returned from a function call and later called in the script or if the object is passed to another method, VuGen uses the same object instance.

You can further correlate values in your script by manually saving the parameters through coding, or through VuGen's built-in correlation tools.

To correlate a value within VuGen, you locate the value in your dataset and use the right-click menu to save the value to a parameter.

To correlate a value for ADO.NET environments:

1 Locate the dataset in your script.

Display the grids in your script to show the returned datasets. If the grids are not visible, select **View > Data Grids** or expand the applicable **DATASET_XML** statement. For example:

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

2 Locate the value.

Locate the value you want to correlate. To search for a value in a grid, open the Find dialog box, Ctrl+F, and select the **Search Grids** option.

3 Create a correlation.

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

4 Specify a parameter name.

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

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

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

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

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

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

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

The above method is effective for ADO.NET environments. For primitive values, you should generate the script with output parameter values and examine the output parameters for correlations.

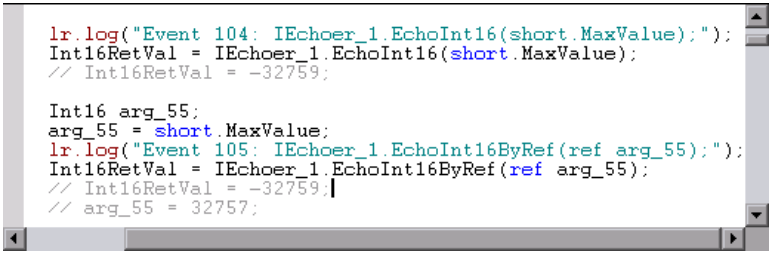
To correlate with output parameters:

1 Select **Tools** > **Recording Options**, and select the **General:Script** node.

2 Enable the **Insert output parameter values** option. Click **OK** to close Recording Options.

3 Select **Tools** > **Regenerate Script** to regenerate the script.

4 Search the commented output primitive values for correlations.



For more information about using correlation functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Configuring Application Security and Permissions

A Security Exception that occurs while recording an application is usually due to a lack of permissions—the recording machine does not have sufficient permissions to record the application. This is common where your application is not local, but on the Intranet or network.

To solve this problem, you need to allow the recording machine to access the application and the script with Full Trust.

One solution is to copy the application and save your script locally, since by default, users have Full Trust permissions to all local applications and folders.

An additional solution is to create new code groups that gives Full Trust to each application folder, and the script folder.

The procedure differs depending on whether you have Visual Studio installed on your machine.

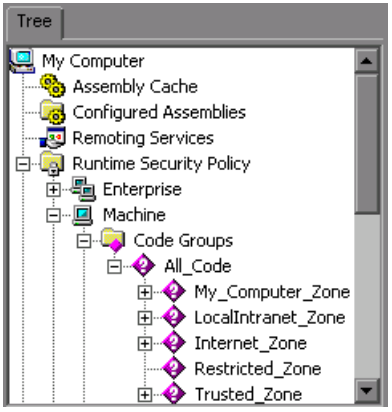
To grant Full Trust permissions to a specific folder if you do not have Visual Studio installed:

1. From the command prompt, run the caspol.exe application.
2. Set the desired permission.

To grant Full Trust permissions to a specific folder if you have Visual Studio installed:

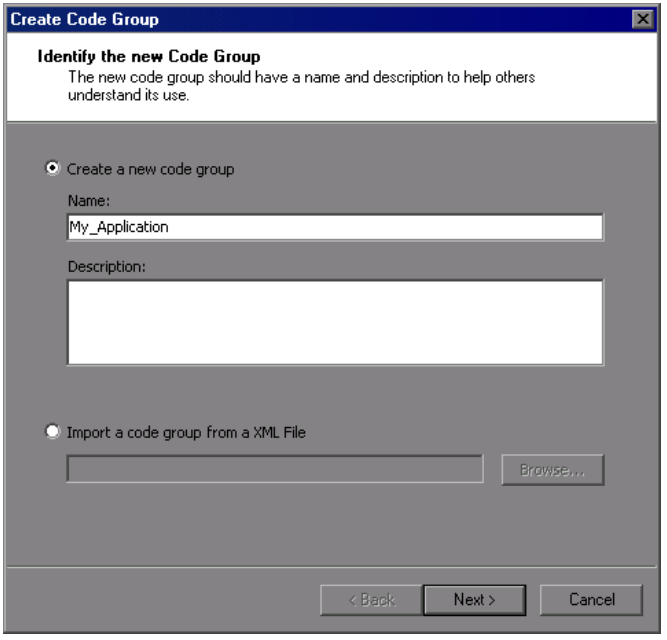
1. Open the .NET Configuration settings. Select **Start** > **Programs** > **Administrative Tools** > **Microsoft .NET Framework 2.0 Configuration**. The .NET Configuration window opens.

2 Expand the **Runtime Security Policy** node to show the Code Groups of the machine.



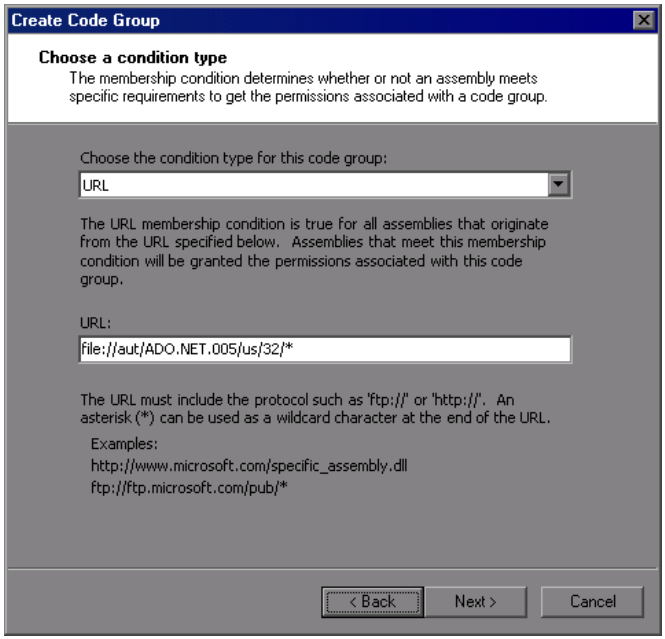
3 Select the **All_Code** node.

4 Select **Action** > **New** …. The Create New Code Group dialog box opens.

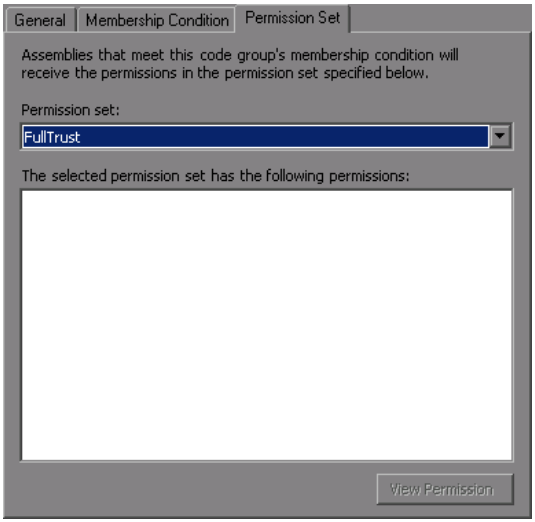


5 Enter a name for a new Code Group for your application or script. Click **Next**.

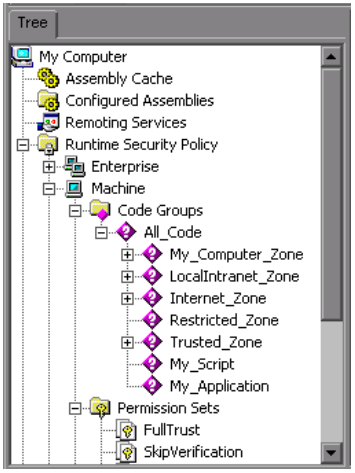
6 Select the **URL** condition type. In the URL box, specify the full path of the application or script in the format file://… and click **Next**.



7 Select the **FullTrust** permission set. Click **Next**.



8 Click **Finish** in the Completing the Wizard dialog box. The configuration tool adds your Code Group to the list of existing groups.



9 Repeat the above procedure for all .NET applications that you plan to record.

10 Repeat the above procedure for the Vuser script folder.

Note: Make sure that the script folder has **FullTrust** permissions on all Load Generator machines that are participating in the test (LoadRunner only).

Recording WCF Duplex Communication

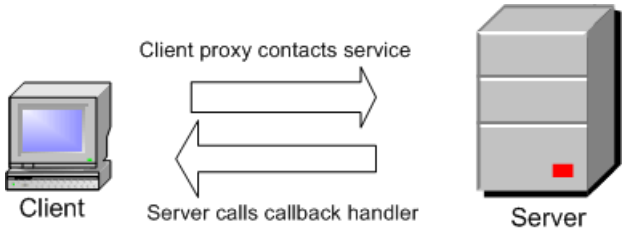
WCF (Windows Communication Foundation) is a programming model that unifies Web Services, .NET Remoting, Distributed Transactions, and Message Queues into a single Service-oriented programming model for distributed computing.

WCF creates a proxy object to provide data for the service. It also marshalls the data returned by the service into the form expected by the caller.

In addition to general support for the WCF environment, VuGen provides specialized support for applications that use WCF's duplex communication. In duplex communication, the client proxy contacts the service, and the service invokes the callback handler on the client machine. The callback handler implements a callback interface defined by the server. The server does not have to respond in a synchronous manner—it independently determines when to respond and invoke the callback handler.

The communication between the client and server is as follows:

- The server defines the service contract and an interface for the callback.
- The client implements the callback interface defined by the server.
- The server calls the callback handler in the client whenever needed.



When trying to record and replay duplex communication, you may encounter problems when the script calls the original callback methods. By default, the callback handlers are not included in the filter. You could customize the filter to include those callback handlers. However, the standard playback would be ineffective for a load test, since many of the callbacks are local operations such as GUI updates. For an effective load test you cannot replay the original callback method invoked by the server.

VuGen's solution is based on replacing the original callback handler with a dummy implementation. This implementation performs a typical set of actions that you can customize further for your application.

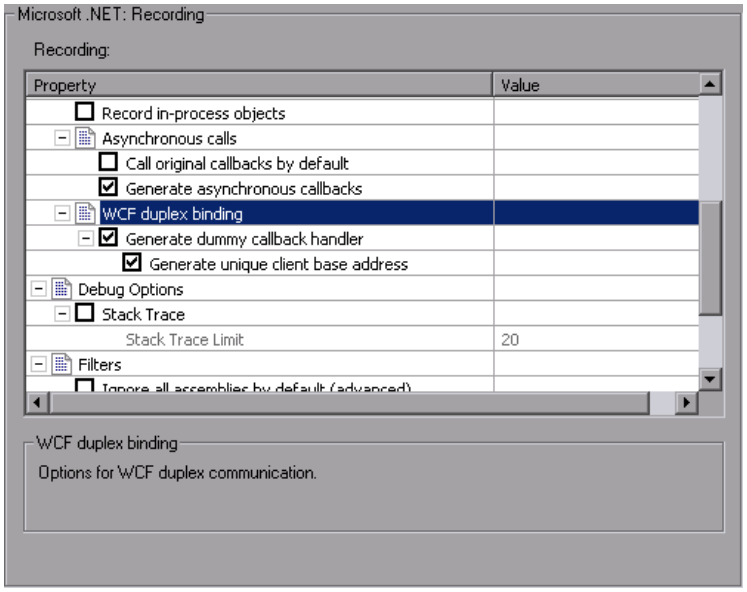
You instruct VuGen to replace the original callbacks by activating the **Generate Dummy Callback Handler** recording option. For more information, see below.

Setting WCF Recording Options

VuGen's recording options for WCF's duplex communications enable you to generate a script that will be effective for load testing. You can set recording options on the following areas:

➤ Generating Dummy Callback Implementations

➤ Recording Dual HTTP Bindings



Generating Dummy Callback Implementations

The **Generate Dummy Callback Handler** recording option instructs VuGen to replace the original callback in duplex communication with a dummy callback.

The dummy callback implementation performs the following actions:

➤ **Store arguments.** When the server calls the handler during replay, it saves the method arguments to a key-value in memory map.

➤ **Synchronize replay.** It stops the script execution until the next response arrives. VuGen places the synchronization at the point that the callback occurred during recording. This is represented in the script by a warning:

```
#warning:  Code Generation Warning
        // Wait here for the next response.
        // The original callback during record was:…
```

As part of the synchronization, the script calls GetNextResponse to get the stored value.

```
Vuser<Callback_Name>.GetNextResponse();
```

Enabling the Dummy Callback Recording Option

By default, this option is enabled.

To enable this recording option:

1 Select **Tools** > **Recording Options**.

2 Select the **Microsoft .NET:Recording** node.

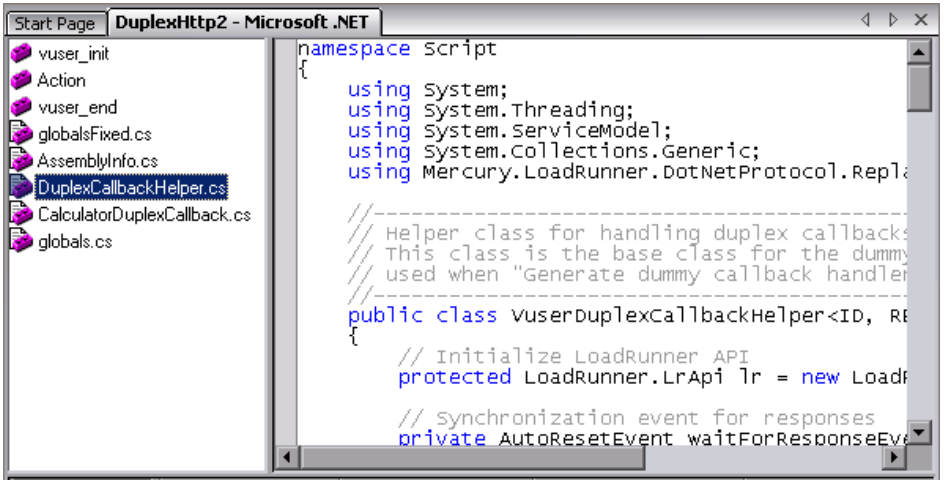
3 Select **Generate Dummy Callback Handler**.

VuGen Implementation of a Duplex Callback

As part of the duplex communication solution, VuGen generates two support files:

- DuplexCallbackHelper.<*language*>
- *Callback_Name*.<*language*>

The following example shows the generated files for a Calculator application using duplex communication:



The Helper file serves as a general template for working with duplex callback handlers. It serves as a base class for the implementation of the callback.

The second file, **Callback_Name**, contains the implementation of the callback. The name of the callback implementation class is **Vuser**<*xxxx*> where *xxxx* is the name of the callback interface and it inherits from the **VuserDuplexCallbackHelper** class defined in the Helper file. VuGen creates separate implementation files for each interface.

This file performs two primary tasks:

- **Set Response.** It stores the data that came from the server in a map. It stores them with sequential IDs facilitating their retrieval. This method is called from the implementation of the callback interface. The following sample code demonstrates the dummy implementation of a callback method named **Result**. The method's arguments are stored in the map as an object array.

```
//  ---------------------------------------------------------------------------
      public virtual void Result(string operation, double result) {
         // Add here your own callback implementation and set the response data
         SetResponse(responseIndex++, new object[] {
                  operation,
                  result});
}
```

- **Get Response.** Waits for the next response to arrive. The implementation of GetNextResponse retrieves the next response stored in the map using a sequential indexer, or waits until the next response arrives.

The script calls GetNextResponse at the point that the original callback handler was called during recording. At that point, the script prints a warning:

```
    // Wait here for the next response.
    // The original callback during record was:
```

Replacement of the Callback in the Script

When you enable the **Dummy Callback** option (enabled by default), VuGen replaces the original duplex callback handlers with dummy implementations. The dummy implementation is called Vuser <*Callback Name*>. At the point of the original callback handler, the script prints a warning indicating that it was replaced.

Recording Dual HTTP Bindings

If your application employs dual HTTP Binding, since HTTP is inherently not a duplex protocol, the framework uses a standard port to receive response data being passed to the callback. When you attempt to run multiple instances of your application, you may be unable to do so using the same port number. VuGen provides you with an option of replacing the original client base address's port number with a unique port.

When you enable the **Generate Unique Client Base Address** recording option, VuGen checks the type of communication used by the application. If it detects dual HTTP communication, **WSDualHttpBinding**, it runs the **FindPort** utility (provided in LrReplayUtils) in the Helper file and finds unique ports for each instance of the callback.

This option is enabled by default. It is only relevant when you enable the above option, **Generate dummy callback handler**.

When you enable this option, VuGen generates the following code in the script:

```
#warning:  Code Generation Warning
    // Override the original client base address with a unique port number
    DualProxyHelper.SetUniqueClientBaseAddress<XXXX>(YYYYY);
```

Customizing the Dummy Implementation

You can modify the implementation file to reflect your environment.

Several suggested customization are:

➤ Timeouts

- Key Identifier
- Return Values
- Get Response Order
- Find Port

Timeouts

The default timeout for which the callback waits for the next response is 60000 msec, or one minute. To use a specific timeout, replace the call to **GetNextResponse** with the overloading method which gets the timeout as an argument as shown below. This method is implemented in the callback implementation file <*Callback_Name*> listed in the left pane after the **DuplexCallbackHelper** file.

```
// Get the next response.
      // This method waits until receiving the response from the server
      // or when the specified timeout is exceeded.
      public virtual object GetNextResponse(int millisecondsTimeout) {
         return base.GetResponse(requestIndex++, millisecondsTimeout);
      }
```

To change the default threshold for all callbacks, modify the **DuplexCallbackHelper** file.

```
// Default timeout threshold while waiting for response
      protected int millisecondsTimeoutThreshold = 60000;
```

Key Identifier

Many applications assign key identifiers to the data, which connects the request and response to one another. This allows you to retrieve the data from the map using its ID instead of the built-in incremental index. To use a key identifier instead of the index, modify the file <*Callback_Name*> replacing the first base template parameter, **named ID**, with the type of your key identifier. For example, if your key identifier is a string you may change the first template argument from **int** to **string**:

```
public class VuserXXX : VuserDuplexCallbackHelper<string, object>
```

In addition, you may remove the implementation of GetNextResponse() and replace it with calls to GetResponse(ID) defined in the base class.

Return Values

By default, since VuGen supports *OneWay* communication, the implementation callback does not return any value or update an output parameter when it is invoked.

```
public virtual void Result(string operation, double result) {
        // Add here your own callback implementation and set the response data
```

If your application requires that the callback return a value, insert your implementation at that point.

Get Response Order

In VuGen's implementation, a blocking method waits for each response. This reflects the order of events as they occurred during recording—the server responded with data. You can modify this behavior to execute without waiting for a response or to implement the blocking only after the completion of the business process.

Find Port

The **FindPort** method in the Helper file is a useful utility that can be used in a variety of implementations. The Helper class uses this method to find unique ports for running multiple instances of the script. You can utilize this utility method for other custom implementations.

Recording Server Hosted By Client Applications

If the communication in your system is a server hosted by a client, VuGen's default solution for duplex communication will not be effective. In server hosted by client environments, it is not true duplex communication since the client opens the service and does not communicate through the Framework. For example, in queuing, the client sends a message to the service and opens a response queue to gather the responses.

To emulate a server hosted by a client, use the pattern depicted in the above solution—replace the original response queues with dummy callbacks and perform synchronization as required. For more information, contact HP support.

47

Microsoft .NET Filters

VuGen provides several built-in filters and also allows you to customize them.

This chapter includes:

About Microsoft .NET Filters

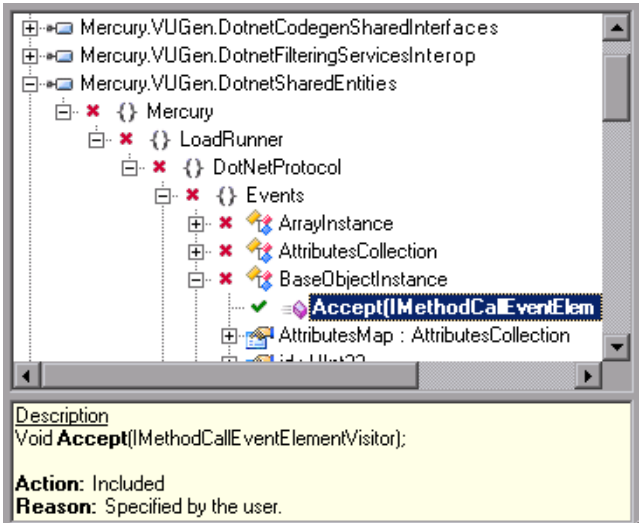
Recording filters indicate which assemblies, interfaces, namespaces, classes, or methods to include or exclude during the recording and script generation.

By default, VuGen provides built-in system filters for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation). These filters were designed to include the relevant interfaces for standard ADO.NET, Remoting, Enterprise Services, and WCF. VuGen also allows you to design custom filters.

Custom filters provide several benefits:

- **Remoting.** When working with .NET Remoting, it is important to include certain classes that allow you to record the arguments passed to the remote method.
- **Missing Objects.** If your recorded script did not record a specific object within your application, you can use a filter to include the missing interface, class or method.
- **Debugging.** If you receive an error, but you are unsure of it's origin, you can use filters to exclude methods, classes, or interfaces in order to pin-point the problematic operation.
- **Maintainability.** You can record script in higher level, make script more easy to maintain and to correlate.

A filter manager lets you manipulate existing custom filters. It displays the assemblies, namespaces, classes, methods, and properties in a color-coded tree hierarchy.



The bottom pane provides a description of the assembly, namespace, class, method, property, or event. It also indicates whether or not it is included or excluded and a reason for the inclusion or exclusion.

The following sections describe when and how to customize a filter:

- Guidelines for Setting Filters
- Setting a Recording Filter
- Working with the Filter Manager

Guidelines for Setting Filters

When testing your .NET application, your goal is determining how the server reacts to requests from the client. When load testing, you want to see how the server responds to a load of many users.

When recording a .NET application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters, .NET Remoting, ADO.NET, Enterprise Services, and WCF, were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your .NET application's calls or exclude unnecessary calls. Using the Filter Manager, you can design custom filters to exclude the irrelevant calls and capture the server related calls.

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, you can use **Visual Studio** or a **Stack Trace** to help you determine which methods are being called by your application in order to include them in the filter. VuGen allows you to record with a stack trace that logs all of the methods that were called by your application.

Once you determine the required methods and classes, you include them using the Filter Manager. When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Tip: Strive to modify the filter so that your script will compile (Shift+F5) inside VuGen—a test with correct syntax. Then customize the filter further to create a functional script that runs inside VuGen.

Note that if you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this is that if you rerecord a script or regenerate the script, you will lose all of the manual changes.

Determining which Elements to Include or Exclude

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- **Top Down Approach.** An approach in which you include the relevant namespace and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined assembly which implements all client-server activity without involving any GUI elements, such as MyDataAccessLayer.dll.
- **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT assemblies and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- If you identify a non-client/server call in your script, exclude its method in the filter.

- During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen **serializes** it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by **deserializing** it.
- VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the **LrReplayUtils.GetSerializedObject** method or, in WCF environments, **LrReplayUtils.GetSerializedDataContract**. VuGen stores serialized objects in the script's **\data\SerializedObjects** directory as XML files with indexes: **Serialization_1.xml**, **Serialization_2.xml** and so forth.
- When no rules are specified for a method, it is excluded by default. However, when the remoting environment is enabled, all remote calls are included by default, even if they are not explicitly included. To change the default behavior, you can add a custom rule to exclude specific calls which are targeted to the remote server.
- Arguments passed in remoting calls whose types are not included by the filter, are handled by the serialization mechanism. To prevent the arguments from being serialized, you can explicitly include such types in order to record the construction and the activity of the arguments.
- Exclude all activity which involves GUI elements.
- Add assemblies for utilities that may be required for the script to be compiled.

For information on how to include and exclude elements, see "Including and Excluding Elements" on page 765.

Defining an Effective Filter

When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

To define an effective filter:

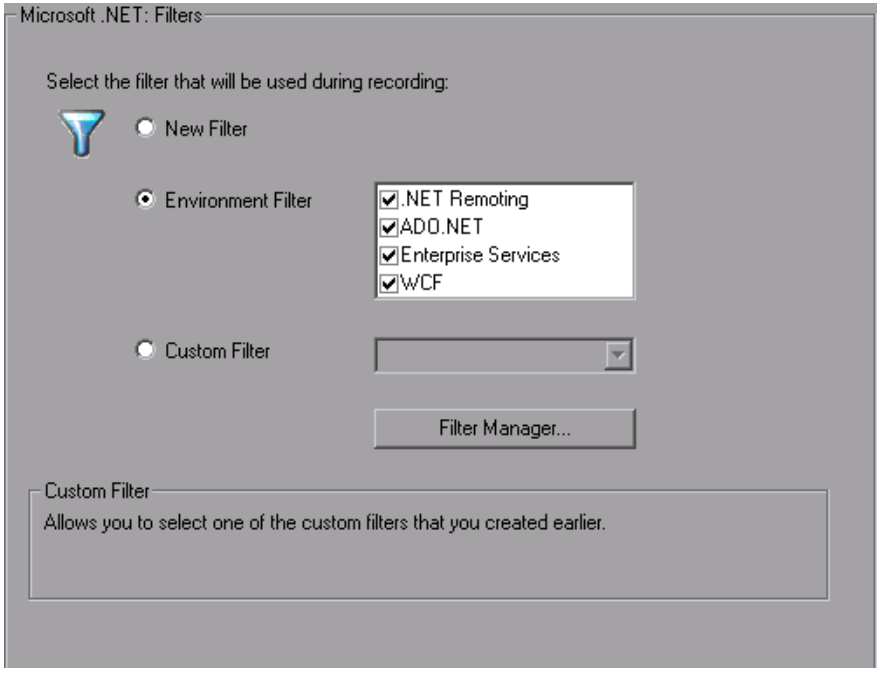
1. Create a new filter based on one of the built-in filters. If you know that the AUT (Application Under Test) does not use ADO.NET, Remoting, WCF, or Enterprise Services, clear that option since unnecessary filters may slow down the recording.
2. Set the **Stack Trace** option to true for both recording and code generation. Open the Recording Options (CTRL+F7) and select the **Recording** node. Enable **Debug Options: Stack Trace** and **Code Generation: Show Stack Trace**.
3. Record your application. Click **Start Record** (CTRL + R) to begin and Stop (CTRL + F5) to end.
4. View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain or correlate, you should customize the script's filter.
5. Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) in Visual Studio or by viewing a Stack Trace of the script.
6. Set the filter to include the relevant methods—you may need to add their assembly beforehand. For tips about including and excluding elements in the filter, see "Determining which Elements to Include or Exclude" on page 756.
7. Record the application again. You should always rerecord the application after modifying the filter.
8. Repeat steps 4 through 7 until you get a simple script which can be easily maintained and correlated.
9. After creating an optimal script, turn off the **Stack Trace** options and regenerate the script. Open the Recording Options (CTRL+F7) and select the **Recording** node. Disable **Debug Options: Stack Trace** and **Code Generation: Show Stack Trace**. This will improve the performance of subsequent recordings.

10 Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information about the built-in correlation mechanism, see "Correlating Microsoft .NET Scripts" on page 736.

Setting a Recording Filter

The built-in filters, .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation), were designed to include the standard interfaces for those environments. For information on the benefits of filters, see "Guidelines for Setting Filters" on page 755.

When you decide to apply a filter to your recording, your first step is choosing an appropriate filter. You can use one or more of the environment filters or create a new one using the **Filters** recording options.



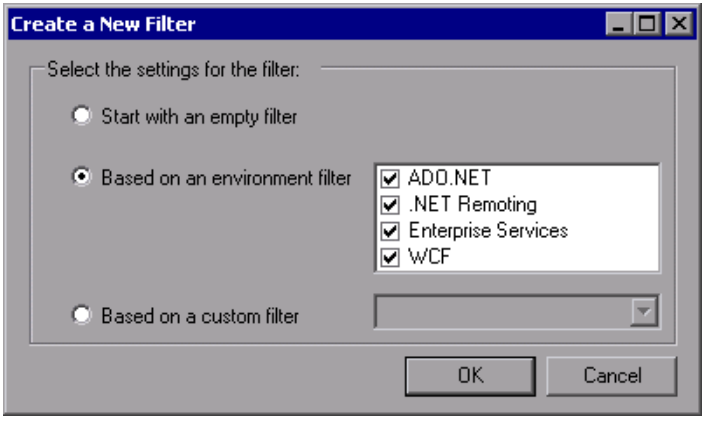
➤ **New Filter.** Indicates that you want to create a new filter.

- **Environment Filter.** Lists the available environment filters: .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation of Framework 3.0).
- **Custom Filter.** Shows the filters that you created earlier on the current machine.

After creating a filter, you can modify the properties of the filter using the Filter Manager. For more information, see "Including and Excluding Elements" on page 765.

To specify a filter:

1. Open the Filters recording option. Select **Tools** > **Recording Options** and select the **Microsoft .NET:Filters** node.
2. Select a filter option: **New Filter**, **Environment Filter,** or **Custom Filter.**
3. For a new filter, click **Create**. The Create a New Filter dialog box opens.



4. Select one of the filter options. To base your new filter on an environment filter, select the check box adjacent to one or more of the filters.
5. Click **OK.** The Filter Manager opens.

   For existing filters, click on the **Filter Manager** button in the main recording options dialog box to open the Filter Manager.

Make the desired modifications to the filters and save the filter. For more information, see below.

Working with the Filter Manager

The Filter Manager lets you view and modify your filters, with the exception of Environment filters which can only be viewed—not modified as they are read-only.

You manage and manipulate your filters using the Filter Manager toolbar.

Managing Filters

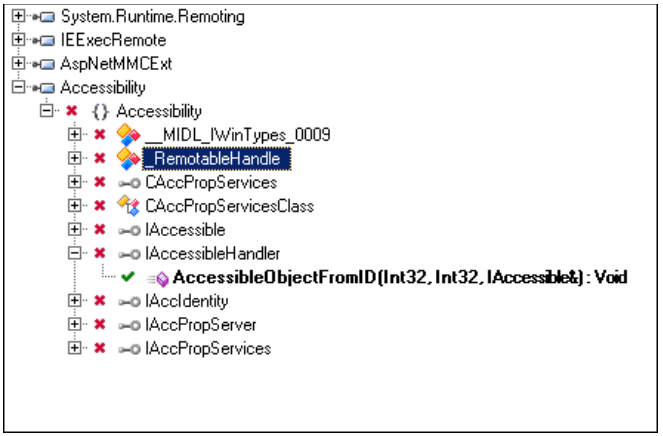
- **New.** Opens the Create a New Filter dialog box, in which you create an empty filter or a new filter based on an existing one
- **Save.** Saves the changes you made to filter.
- **Delete.** Deletes the selected custom filter. The Filter Manager prompts you for a confirmation.

Manipulating Filters

- **Add Reference.** Opens the Add Reference dialog box with a list of .NET Framework components or assemblies in the Public Assemblies folder. You can also browse the computer to locate a component that is not on the list. For more information, see "Adding References" on page 763.
- **Remove Reference.** Removes the assembly that is selected in the Filter Manager and all of the elements associated with it. The Filter Manager prompts you for a confirmation.
- **Include**, **Exclude**, and **Reset**. Includes or excludes an assembly, namespace, class, or method. You can also reset the inclusion or exclusion rule to its default state. For more information, see "Including and Excluding Elements" on page 765.
- **Back** and **Forward**. Navigates to the previous or next tree node visited by the user.
- **View Impact Log.** Opens the Impact log for the selected filter. The Impact log shows which nodes in the tree were affected by recent actions. For more information, see "Viewing an Impact Log" on page 767.

In addition, you can copy, paste, and rename filters using the standard Windows key combinations and right-click menu.

The Filter Manager tree uses symbols to illustrate the elements and their status:



- Element icons represent the type of element—assembly, namespace, class, method, structure, property, events, or interfaces.
- A check mark or X adjacent to the element icon, indicates whether or not the element is included or excluded.
- A bolded element indicates that it was explicitly included or excluded. This may be a result of being manually included or excluded by the user or by a pre-defined rule in the environment filter. If you reset a bolded node, it returns to its original, unbolded state.

The following table shows the icons that represent the various elements.

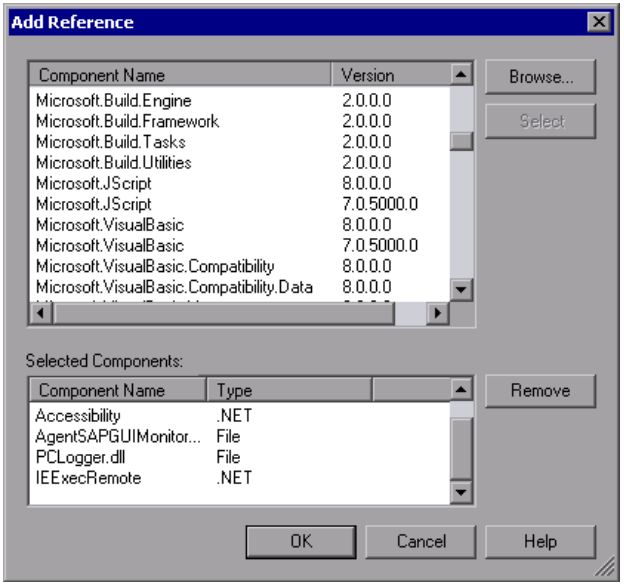
	assembly
	assembly that couldn't be loaded
	assembly that was partially loaded
	class
	constructor
	static constructor
	event
	static event

	interface
	method
	static method
	namespace
	property
	static property
	structure

Adding References

When you create a new filter, you must add assembly references to the filter to indicate its behavior. In addition, you can add references to pre-existing custom filters.

When you open the Add Reference dialog box, the Filter Manager lists all of the public assemblies in the GAC (Global Assembly Cache). You can add references that are not listed using the **Browse** button.



In the bottom pane, the **Type** column differentiates between the references. References that reside in the GAC are of the **.NET** type and references that do not reside in the GAC are of **File** type.

The list of references may include different versions of the same assembly. In this case, select the version that is most appropriate for your test.

To add a reference through the Filter Manager:

1. Click the **Add Reference** button on the toolbar. The Add Reference dialog box opens.
2. To add one of the listed items, select it and click **Select**. You can select multiple components using CTRL-CLICK. The bottom pane shows the selected references.
3. To add an assembly that is not in the list, click **Browse** and locate the reference on your file system or network.

4 Repeat the above steps for all of the references you want to add to your filter.

5 Review the list of references displayed in the bottom pane. To clear an item from the list, select it in the bottom pane and click **Remove**.

6 When you finish creating your list of references, click **OK** to close the dialog box and add the references to the filter. VuGen adds it to the end of the list of elements in the Filter Manager's tree. If any of the references you chose are not valid assemblies, VuGen will issue an error message.

7 To remove a reference from the Filter Manager's tree, select the parent assembly node and select **Remove Reference** from the right-click menu, or click the Remove Reference button on the toolbar.

After you add a reference, you should view it in the Filter Manager and determine if all the correct nodes are included or excluded. If necessary, you can include or exclude specific namespaces, classes, or methods. For more information, see below, Including and Excluding Elements.

Including and Excluding Elements

After you add references to the filter, you can view all of its nodes in the Filter Manager's tree. You can exclude specific namespaces, classes, or methods, or include those that were excluded by default or by other rules. The description in the Filter Manager's lower pane, indicates the reason for the inclusion or exclusion of the element.

The Filter Manager's toolbar provides the following buttons for including or excluding elements:

➤ **Include.** Indicated by a check mark, includes the selected element. If you manually include a parent node, the Filter Manager includes the child elements below it, provided that no other rule exists. For example, if you include a class, it will include all its methods unless you specifically excluded a method.

- **Exclude.** Indicated by an X, excludes the selected element. The child elements are also excluded unless they were included by another rule. By default, when you exclude a **class**, the Filter Manager applies the **Exclude** attribute to the class, but it allows the recording engine to record activity within the methods of the excluded class. When you exclude a **method**, however, the Filter Manager applies **Totally Exclude**, preventing the recording engine from recording any activity within the methods of the excluded class. Advanced users can modify these setting in the filter file. For more information, see "Advanced Information About Filter Files" on page 768.

- **Reset.** Removes the manual inclusion or exclusion rule. In this case, the element may be impacted by other parent elements.

The inclusion and exclusion rules have the following properties:

- The rules are hierarchical—if you add an include or exclude rule to a class, then the derived classes will follow the same rule unless otherwise specified.
- A rule on a class only affects its public methods, derived classes, and inner classes.
- A rule on a namespace affects all the classes and their public methods.
- Note that adding or removing assemblies does not necessarily affect the classes that they contain—you can remove an assembly, yet its methods may be recorded due to the hierarchical nature of the filter.
- As part of the filter design, several methods, such as **.cctor()** and **Dispose(bool)**, do not follow the standard hierarchal rules.

Note: The resetting of a parent node does not override a manual inclusion or exclusion applied to a child node. For example, if you manually **exclude** a method, and then reset its class, which by default **included** all sub-nodes, your method will remain excluded.

Properties and events are view-only and cannot be included or excluded through the Filter Manager. In addition, several system related elements are protected and may not be altered.

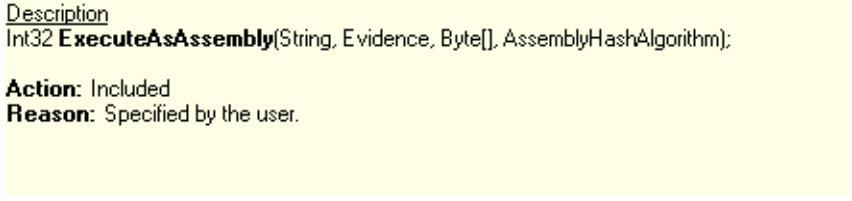
For tips about including and excluding elements in the filter, see "Determining which Elements to Include or Exclude" on page 756.

To add or remove assemblies, use the **Add** and **Remove** Reference buttons as described in "Adding References" on page 763. The following section describes how to include namespaces, classes, and methods.

To include or exclude an element:

1. Expand the tree hierarchy and select a namespace, class, or method.
2. To include an element, select it and click the **Include** button or use the Include command on the right-click menu.
3. To exclude an element, click the **Exclude** button or use the Exclude command on the right-click menu.
4. To reset an element to its default settings, click the **Reset** button or select **Reset** from the right-click menu.

To verify that the change took effect, select the component and view the bottom pane.

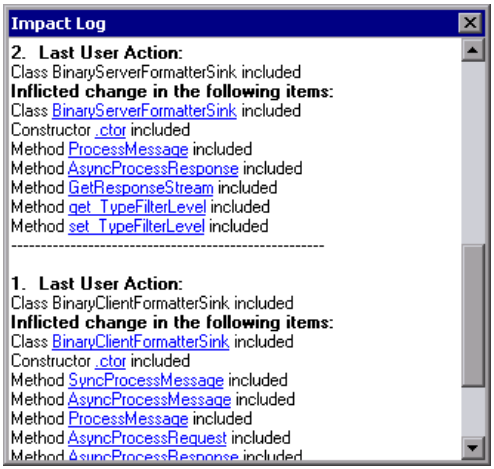


Viewing an Impact Log

The Impact Log indicates what your last changes were and how they affected your filter. The user actions are listed in descending order, with the latest changes at the top.

For each element affected by your manual inclusion or exclusion, the log indicates how it affected the element. It also provides a link to that element in the Filter Manager.

To view the Impact Log, click the Impact Log button on the Filter Manager's toolbar or select **Actions** > **View Impact Log** in the Filter Manager window.



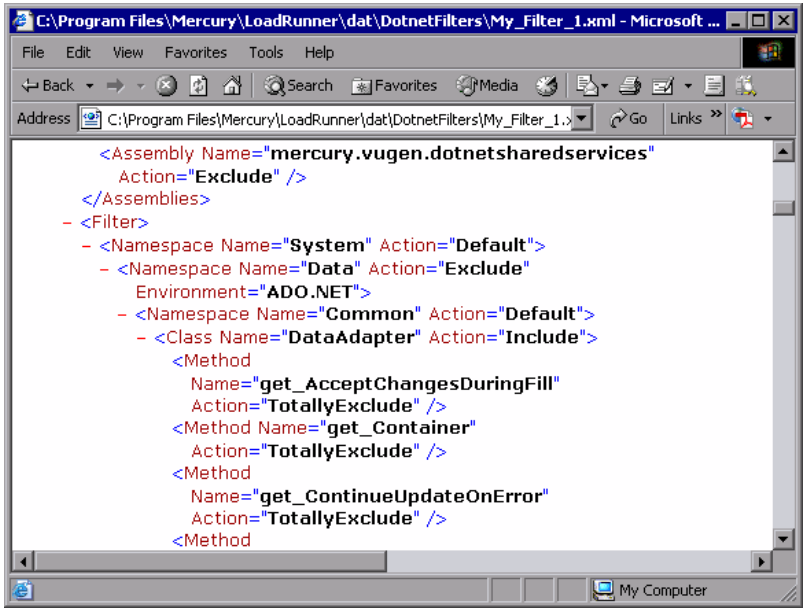
Advanced Information About Filter Files

In the Filter Manager's tree hierarchy, it only displays public classes and methods. It does not show non-public classes or delegates.

You can add classes or methods that are not public by manually entering them in the filter's definition file.

The filter definition files, **<filter_name>.xml** reside in the dat\DotnetFilters folder of your installation. The available Action properties for each element are: **Include**, **Exclude**, or **Totally Exclude**. For more information, see "Including and Excluding Elements" on page 765.

By default, when you exclude a **class**, the Filter Manager applies **Exclude**, excluding the class, but including activity generated by the excluded class. When you exclude a **method**, however, it applies **Totally Exclude**, excluding all referenced methods.



For example, suppose Function A calls function B. If Function A is **Excluded**, then when the service calls Function A, the script will include a call to Function B. However, if function A is **Totally Excluded**, the script will not include a call to Function B. Function B would only be recorded if called directly—not through Function A.

VuGen saves a backup copy of the filter as it was configured during the recording, **RecordingFilterFile.xml**, in the script's **data** folder. This is useful if you made changes to the filter since your last recording and you need to reconstruct the environment.

48

Web (HTTP/HTML, Click and Script) Protocols

You use VuGen to develop Web Vuser scripts based on your actions while you operate a client browser.

This chapter includes:

About Developing Web Level Vuser Scripts

You use VuGen to develop Web Vuser scripts. While you navigate through a site performing typical user activities, VuGen records your actions and generates a Vuser script. When you run the script, the resulting Vuser emulates a user accessing the Internet.

After you create a Vuser script, you run the script in stand-alone mode using VuGen. When the execution is successful, you are ready to integrate the Vuser script into a scenario. For details on how to integrate a Vuser script into a scenario, see the *HP LoadRunner Controller User's Guide*.

For certain Vuser types, you can create a Business Process Report for Microsoft Word that provides information about the script and the events that were recorded. For more information, see Chapter 7, "Creating Business Process Reports."

Introducing Web Vusers

Suppose you have a Web site that displays product information for your company. The site is accessed by potential customers. You want to make sure that the response time for any customer query is less than a specified value (for example, 20 seconds)—even when a large number of users (for example, 200) access the site simultaneously. You use Vusers to emulate this case, where the Web server services simultaneous requests for information. Each Vuser could do the following:

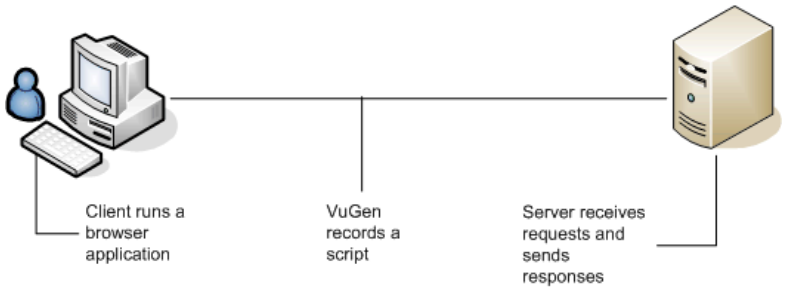
- Load a home page
- Navigate to the page containing the product information
- Submit a query
- Wait for a response from the server

You can distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

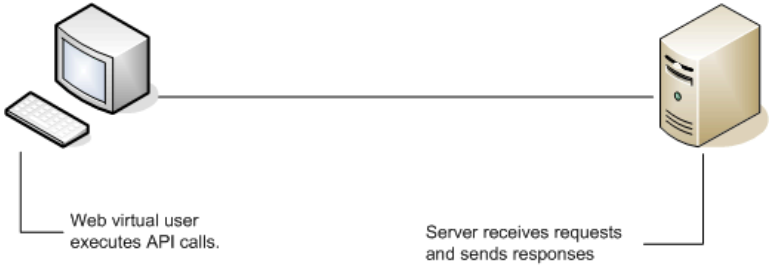
The program that contains the calls to the server API is called a Vuser script. It emulates a browser application and all of the actions performed by the browser. Using the Controller, you assign the script to multiple Vusers. The Vusers execute the script and emulate user load on the Web server.

Understanding Web Vuser Technology

VuGen creates Web Vuser scripts by recording the activity between a browser and a Web server. VuGen monitors the client (browser) end of the system and traces all the requests sent to, and received from, the server.



When you run a recorded Vuser script, the Vuser communicates directly with the server without relying on client software. Instead, the Vuser script executes calls directly to the Web server via API functions.



Selecting a Web Vuser Type

When creating a new Web Vuser script, you can select one of the following types of Web Vusers:

- Web (Click and Script)
- Web (HTTP/HTML)

Web (Click and Script)

The Web (Click and Script) Vuser is a solution for recording Web sessions on a user-action GUI level. VuGen creates a GUI-level script that intuitively represents actions in the Web interface. For example, it generates a **web_button** function when you click a button to submit information, and generates a **web_edit_field** function when you enter text into an edit box.

Web (Click and Script) Vusers support non-HTML code such as Javascript on the client side. VuGen creates an intuitive script that accurately emulates your actions on the Web page and executes the necessary Javascript code.

Web (Click and Script) Vusers handle most correlations automatically, reducing the scripting time. In most cases, you do not need to define rules for correlations or perform manual correlations after the recording.

Web (Click and Script) Vusers also allow you to generate detailed Business Process Reports which summarize the script.

For example, when you click a button to submit data, VuGen generates **web_button**. If the button is an image, VuGen generates **web_image_submit**. In the following example, a user clicked the login button.

```
…
web_image_submit("Login",
        "Snapshot=t4.inf",
        DESCRIPTION,
        "Alt=Login",
        "Name=login",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=31,6",
        LAST);}
```

The next section illustrates a user navigating to the Asset ExpressAdd process under the Manage Assets branch. The user navigates by clicking the text links of the desired branches, generating **web_text_link** functions.

```
web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Ordinal=2",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);

web_text_link("Use",
        DESCRIPTION,
        "Text=Use",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);

web_text_link("Asset ExpressAdd",
        DESCRIPTION,
        "Text=Asset ExpressAdd",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

In the following example, **web_list** emulates the selection of a list item.

```
…
web_list("Year",
        DESCRIPTION,
        "Name=Year",
        "FrameName=CalFrame",
        ACTION,
        "Select=2000",
        LAST);
```

When you click on an image that is associated with an image map, VuGen generates a **web_map_area** function.

```
web_map_area("map2_2",
        DESCRIPTION,
        "MapName=map2",
        "Ordinal=20",
        "FrameName=CalFrame",
        ACTION,
        "UserAction=Click",
        LAST);
```

Note: Web (Click and Script) Vusers do not support Applets or VB Script. If the Web site under test contains these items, use the Web (HTTP/HTML) user.

Web (HTTP/HTML)

When recording a Web (HTTP/HTML) script, VuGen records the HTTP traffic between the browser and the server. The scripts contain detailed information about the recorded traffic.

The Web (HTTP/HTML) Vuser provides two recording levels: **HTML-based script** and **URL-based script**. These levels lets you specify what information to record and which functions to use when generating a Vuser script. For more information about selecting a Recording level, see "Understanding the Recording Levels" on page 1218.

Tip: For most applications, including those with JavaScript, use Web (Click and Script) Vusers. For browser applications with applets and VB Script or for non-browser applications, use the Web (HTTP/HTML) Vuser.

Getting Started with Web Vuser Scripts

This section provides an overview of the process of developing Web Vuser scripts.

To develop a Web Vuser script:

1 Create a new script using VuGen.

Click on VuGen's **Start Page** tab and click on **File** > **New**. Select Web (Click and Script) or Web (HTTP/HTML) Vuser script from the **e-business** category, in either single or multiple protocol mode.

For details about creating a new script, see Chapter 5, "Recording with VuGen."

2 Set the recording options.

Set the recording options. For information about setting common Internet recording options, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168.

For details about selecting a recording level, see Chapter 77, "Setting Recording Options for Web Vusers."

For Web (Click and Script) specific options, see Chapter 74, "Click and Script Recording."

3 Record a browser session.

Record your actions while you navigate your Web site.

For details about creating a new script, see Chapter 5, "Recording with VuGen."

4 Enhance the recorded Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, checks, and service steps.

For details, see Chapter 51, "Web (HTTP/HTML, Click and Script) Text and Image Verification", Chapter 52, "Modifying Web and Wireless Vuser Scripts", and Chapter 53, "Web (HTTP/HTML) Correlation Rules."

5 Define parameters (optional).

Define parameters for the fixed values recorded into your script. By substituting fixed values with parameters, you can repeat the same Vuser action many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

6 Configure the run-time settings.

The run-time settings control Vuser behavior during script execution. These settings include general run-time settings (iteration, log, think time, and general information), and Web-related settings (proxy, network, and HTTP details).

For details, see Chapter 79, "Configuring Run-Time Settings."

7 Perform correlation.

For Web (HTTP/HTML) scripts, Scan your Vuser script for correlations and use one of VuGen's mechanisms to implement them.

For details on setting up automatic correlation, see Chapter 53, "Web (HTTP/HTML) Correlation Rules." For information on correlation after the recording, see Chapter 54, "Web (HTTP/HTML) -Correlation After Recording."

8 Run and debug the Vuser script using VuGen.

Run the Vuser script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode" and Chapter 10, "Viewing Test Results."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Recording a Web Session

When you record a Web session, VuGen monitors all the actions that you perform in your Web browser. Your activities can include hyperlink jumps (both hypertext and hypergraphic) and form submissions. While recording, VuGen saves the recorded actions in a Web Vuser script.

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. The vuser_init and vuser_end sections are generally used for recording server logon and logoff procedures, which are not repeated when you run a Vuser script with multiple iterations.

You should therefore record a Web session into the Actions sections so that the complete browser session is repeated for each iteration.

Converting Web Vuser Scripts into Java

VuGen provides a utility that enables you to convert a script created for a Web Vuser into a script for Java Vusers. This also allows you to create a hybrid Vuser script for both Web and Java.

To convert a Web Vuser script into a Java Vuser script:

1. Create an empty Java Vuser script and save it.
2. Create an empty Web Vuser script and save it.
3. Record a Web session using standard HTML/HTTP recording.
4. Replay the Web Vuser script. When it replays correctly, cut and paste the entire script into a text document and save it as a text **.txt** file. In the text file, modify any parameter braces from the Web type, "{ }" to the Java type, "< >".
5. Open a DOS command window and go to your product's **dat** directory.
6. Type the following command:

```
<application_directory>\bin\sed -f web_to_java.sed filename > outputfilename
```

   where **filename** is the full path and filename of the text file you saved earlier, and **outputfilename** is the full path and filename of the output file.

7. Open the output file, and copy its contents into your Java Vuser script action section at the desired location. If you are pasting the contents into an empty custom Java template (Java Vuser type), modify the line containing `public int action()` as follows:

```
public int action() throws Throwable
```

   This change is done automatically for recorded Java users (RMI and CORBA).

Parameterize and correlate the Vuser script as you would with an ordinary Java script and run it.

Support for Push Technology

Vuser scripts run steps sequentially, one step cannot start until the previous step has completed. A step does not complete until all requested data has finished downloading. Some websites use push technology in which downloads are not meant to complete. Data is periodically downloaded as updates occur.

When a Vuser script contains steps which access resources using push technology, the Vuser script gets stuck on the step until a timeout occurs. This is a "false" timeout because the site is functioning as designed.

Expert users can work around this problem by changing the conditions for the step to complete. For more information, see the *Online Function Reference* (**Help > Function Reference**).

49

Web (Click and Script) Tips

The Record, Replay and Enhancement tips provide guidelines for developing Web (Click and Script) scripts.

This chapter includes:

The following information applies to the Web (Click and Script), AJAX (Click and Script), SAP (Click and Script), Oracle Web Applications 11i, and PeopleSoft Enterprise protocols.

Recording Issues

The following section lists the most common recording problems:

Firefox is not supported

Only Internet Explorer is supported for Web (Click and Script). To record browser activity on Firefox, use the Web (HTTP/HTML) protocol.

Application behaves differently while being recorded

If your application behaves differently during recording, than it does without recording, you should determine if the recording problem is unique to Web (Click and Script). The effect may be that a Web page will not load, part of the content is missing, a popup window does not open, and so forth.

Create a new Web (HTTP/HTML) script and repeat the recording.

In the event that the recording fails in Web (HTTP/HTML), we recommend that you disable socket level recording (see "Disable socket level recording" on page 786).

The problem may be the result of an event listener. Use trial and error to disable event listeners in the **Web Event Configuration** Recording Options, and then re-record your session as a Web (Click and Script) user.

To disable an event listener:

➤ Open the Recording Options. Select **Tools** > **Recording Options** and select the **GUI Properties:Web Event Configuration** node.

➤ Click **Custom Settings** and expand the **Web Objects** node. Select an object.

➤ Select **Disabled** from the list in the **Record** column for the relevant Web object. If the recording still does not work, enable the listener you previously disabled, and try disabling another one. Repeat these steps until your recording succeeds.

Dynamic menu navigation was not recorded

A dynamic menu is a menu that dynamically changes depending on where you select it. If the dynamic menu navigation was not recorded, record again using “high” event configuration mode.

To set the configuration level to high:

➤ Open the Recording Options. Select **Tools** > **Recording Options** and select the **GUI Properties:Web Event Configuration** node.

➤ Move the slider to **High**.

Certain user actions were not recorded

Check if there is a Java applet running inside the browser. If not, record the script with the Web (HTTP/HTML) protocol.

Recording Tips

Use the mouse and not the keyboard

It is preferable to click on an object with the mouse rather then using the keyboard. During recording, use only GUI objects that are within the browser's pane. Do not use any browser icons, controls, the Stop button, or menu items, such as **View > Refresh**. You may, however, use the Refresh, Home, Back and Forward buttons and the address bar.

Do not record over an existing script

It is best to record into a newly created script—not an existing one.

Avoid context menus

Avoid using context menus during recording. Context menus are menus which pop up when clicking an item in a graphical user interface, such as right-click menus.

Avoid working in another browser while recording

During recording, do not work in any browser window other than the browser windows opened by VuGen.

Wait for downloads

Wait for all downloads to complete before doing any action, such as clicking on a button or filling in a text field.

Wait for pages to load

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Navigate to start page

If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page at the end of your recording, so that the same link will be visible at the end of the business process.

Use a higher event configuration level

Record the business process again the **High** Event Configuration level. For more information on changing the Event Configuration level, see "Dynamic menu navigation was not recorded" on page 784.

Disable socket level recording

In certain cases, the capturing of the socket level messages disrupts the application. For most recordings, socket level data is not required. To prevent the recording of socket level data, disable the option in the recording options. For more information, see the section about recording with Click and Script.

Enable the record rendering-related property values

If the client-side scripts of the application use a lot of styling activities, enable the **Record rendering-related property values** option before recording the script. For example, enable this option to record additional DOM properties such as **offsetTop**. Note that enabling this option may decrease the recording speed.

To enable record rendering-related property values:

➤ Open the Recording Options. Select **Tools** > **Recording Options** and select the **GUI Properties:Advanced** node.

Replay Problems

GUI object not found

Does the error occur at the beginning of the second iteration?

If the error occurs at the beginning of the second iteration's Action section, it is probably the result of a starting page that was present for the first iteration, but missing for the second one. If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page, so that the same link will be visible at the end of the business process.

Is it a text link containing non-ASCII characters?

If the problem occurs with non-ASCII characters, you should instruct VuGen to covert the data to a suitable character set.

To enable data conversion on Windows machines:

1. Open the Run-Time settings. Select **Vuser** > **Run-Time Settings** and select the **Internet Protocol:Preferences** node.
2. Click **Options** to open the Advanced Options dialog box.
3. Locate **Charset Conversions by HTTP** in the Web (Click and Script) > General options, and set it to **Yes**.

To enable UTF-8 conversion for UNIX machines:

1. Open the Run-Time settings. Select **Vuser** > **Run-Time Settings** and select the **Internet Protocol:Preferences** node.
2. Click **Options** to open the Advanced Options dialog box.
3. Locate **Convert from/to UTF-8** in the General options and set it to **Yes**.

Alternatively, view the list of alternatives that are displayed when a link is not found. Enter the displayed text as-is, such as hex escape sequences \xA0 or any other non-standard format.

Can you run the same sequence of actions twice in the application?

In some cases, you can only perform a certain process once, such as deleting a user from the database. Replay will fail after the first iteration, because the action is no longer valid. Verify that your business process can be repeated in the application more than once with the same data, without recording again.

Were the image properties 'Id', 'Name' and 'Alt' empty?

In Tree view, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Did the step's description change?

Check the Replay Log in the Output window, for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the Script View and manually modify the value of the step's DESCRIPTION argument(s).
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument(s). For more information, see the *Online Function Reference* (**Help** > **Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Did the page load completely during recording?

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Replay Tips

The following tips may help you in troubleshooting your problems:

Do not reorder

Do not reorder the statements within a recorded script. Also, copying segments of code from one Action to another is not recommended.

Convert non-ASCII characters

If your links contain non-ASCII characters, you should instruct VuGen to convert the data to or from the UTF-8 format.

To enable UTF-8 conversion:

➤ Open the Recording Options. Select **Vuser** > **Run-Time Settings** and select the **Internet Protocol:Preferences** node.

➤ Click **Options** to open the Advanced Options dialog box.

➤ Locate the **Convert from/to UTF-8** option and set it to **Yes**.

Alternatively, view the list of alternatives that are displayed when a link is not found. Enter the displayed text as-is, such as the hex escape sequences \xA0 or any other non-standard format.

Run same sequence of actions twice

In some cases, you can only perform a certain process once, such as deleting a user from the database. Replay will fail after the first iteration, because the action in no longer valid. Verify that your business process can be repeated in the application more than once with the same data, without recording.

Set unique image properties

In Tree view, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Check the step's description

If you receive an error **GUI Object is not found**, check the Replay Log in the Output window, for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the Script View and manually modify the value of the step's DESCRIPTION argument.
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Miscellaneous Problems

Out of memory error in JavaScript

Increase the JavaScript memory in the Run-Time settings.

To increase the JavaScript memory size:

1. Open the Recording Options. Select **Vuser** > **Run-Time Settings** and select the **Internet Protocol:Preferences** node.
2. Click **Options** to open the Advanced Options dialog box.
3. Locate the **Memory Management JavaScript Runtime Memory Size (Kb)** and **Memory Management JavaScript Stack Memory Size (Kb)** options.
4. Increase the memory sizes to 512 or higher.

VuGen displays JavaScript errors

If VuGen displays JavaScript errors in the Replay Log, verify that the Javascript itself does not contain errors, by enabling IE (Internet Explorer) script errors.

To show script errors:

1. Open Internet Explorer. Select **Tools** > **Internet Options** and select the **Advanced** tab.
2. Enable the **Display a notification about every script error** under the **Browsing** section.
3. Rerun the application in IE. If IE displays script errors, then there is a problem with the JavaScript application. If it is not possible to fix the application, you can safely ignore the corresponding replay errors.

Problems following parameterization

If you encounter problems only after you have parameterized values, verify that the values are valid for your application. Perform business process with the value of the parameter and verify that the application accepts it.

Problems with applications that utilize styling actions

If the client-side scripts of the application use a lot of styling activities, you should record the script again after enabling the **Record rendering-related property values** option. This enables the recording of additional DOM objects.

To enable record rendering-related property values:

1. Open the Recording Options. Select **Tools** > **Recording Options** and select the **GUI Properties:Advanced** node.
2. Enable the **Record rendering-related property values** option. Re-record the script.

Miscellaneous Tips

The following additional tips may help you in troubleshooting your problems:

Search for warnings

Search for warnings or alerts in the Replay Log.

Verify the response

Verify the response of the previous step is correct using **web_reg_find**. For more information, see the *Online Function Reference* (**Help > Function Reference)**.

Use alternate navigation

For problematic steps or those using Java applets, Use **Alternative Navigation** to replace the Web (Click and Script) step with an HTTP level step. Note that the HTTP level steps may require manual correlations. To perform Alternative Navigation, select a step in Tree View, or the text in Script View, and select **Replace with alternative navigation** from the right-click menu.

Working with the Kerberos Protocol

If you are using the Kerberos Protocol for authentication, you must customize VuGen to properly convene authorization sessions. Advanced users can attempt to perform this customization themselves.

In order for the Kerberos Protocol to work properly, create a krb5.ini file and put it in an available directory. Save the full pathname of krb5.ini into the KRB5_CONFIG envoronment variable.

The krb5.ini file should contain detailed information about each domain (KDS and AS addresses) and trust chains.

For more information, contact HP software support.

Enhancing Your Web (Click and Script) Vuser Script

The following section describes several enhancements that can assist you in creating your script.

Most of the features described below are enhancements to the API functions. For detailed information about the functions and their arguments, see the *Online Function Reference* (**Help > Function Reference**) or click F1 on any function.

Adding conditional steps

The Web (Click and Script) functions, **web_xxxx**, allow you to specify conditional actions during replay. Conditions are useful, for example, if you need to check for an element and perform an action only if the element is found.

For example, suppose you perform an Internet search and you want to navigate to all of the result pages by clicking Next. Since you do not know how many result pages there will be, you need to check if there is a Next button, indicating another page, without failing the step. The following code adds a verification step with a notification—if it finds the Next button, it clicks on it.

```
While (web_text_link("Next",
DESCRIPTION,
    "Text=Next",
    VERIFICATION,
    "NotFound=Notify",
    ACTION,
    "UserAction=Click",
    LAST) == LR_PASS);
```

For details about the syntax and use of the VERIFICATION section, see the *Online Function Reference* (**Help > Function Reference**).

Checking a page title

In **web_browser** steps, you can use the title verification recording option to make sure that the correct page is downloaded. You can instruct the Vuser to perform this check automatically for every step or every navigation to a new top level window.

In addition, you can manually add title verifications to your script at the desired locations, using both exact and regular expression matches.

```
web_browser("test_step",
DESCRIPTION,
…
VERIFICATION,
   "BrowserTitle=Title",
   ACTION,]
,
LAST);
```

For more information, see the *Online Function Reference* (**Help > Function Reference**).

You can set title verification options directly from within the Recording options. For more information, see see the section about recording with Click and Script.

Text check verification

Using text checkpoints, you can verify that a text string is displayed in the appropriate place on a Web page or application and then perform an action based on the findings. You can check that a text string exists (**ContainsText**), or that it does not exist (**DoesNotContainText**), using exact or regular expression matching.

For example, suppose a Web page displays the sentence "Flight departing from New York to San Francisco". You can create a text checkpoint that checks that the words "New York" are displayed between "Flight departing from" and "to San Francisco". (In this example, you would need to use regular expression criteria.)

To implement these checkpoints, you add the Text Check related arguments to the VERIFICATION section of the step. During replay, Vusers search the innerText of the browser's HTML document and any child frames. The **NotFound** argument specifies the action to take if verification fails, either because the object was not found or because the text verification failed: Error, Warning, or Notify.

You can manually add text verifications to your script for existing steps. Place the text verification after the step that generated the element.

The text validation arguments are valid for the following Action functions: **web_browser**, **web_element**, **web_list**, **web_text_link**, **web_table**, and **web_text_area**.

Note: You can only use the same type of text verification once per step (for example, **ContainsText** twice). If you want to check for multiple texts, separate them into several steps. You can, however, use different verifications in the same step (for example, **ContainsText** & **DoesNotContainText**). In this case, all conditions have to be met in order for the step to pass.

In the following example, the verification arguments check that we were not directed from www.acme.com to the French version of the website, acme.com/fr.

```
web_browser("www.acme.com",
        ACTION,
        "Navigate=http://www.acme.com/",
    LAST);

web_browser("Verify",
        VERIFICATION,
        "ContainsText=Go to Acme France",
        "DoesNotContainText=acme.com in English",
    LAST);
```

Saving a Java script value to a parameter

The **EvalJavaScript** argument lets you evaluate Java Script on the Web page.

Suppose you want to click on a link which has the same name as the page title. The following example evaluates the document title and uses it in the next web_text_link function.

```
web_browser("GetTitle",
    ACTION,
    "EvalJavaScript=document.title;",
    "EvalJavaScriptResultParam=title",
    LAST);

web_text_link("Link",
    DESCRIPTION,
    "Text={title}",
    LAST);
```

Working with custom descriptions

Suppose you want to randomly click a link that belongs to some group. For example, on **hp.com** you want to randomly select a country. Regular description matching will not allow this type of operation. However, using a custom description argument, you can identify the group with an attribute that is common to all the links in the group.

Using the custom description argument, you specify any attribute of the element, even those that are not predefined for that element. During replay, the Vuser searches for those attributes specified in the DESCRIPTION section. Replay will not fail on any unknown argument in the DESCRIPTION section.

For example, to find the following hyperlink:

Yahoo, use:

```
web_text_link("yahoo",
    DESCRIPTION,
    "Text=yahoo",
    "my_attribute=bar",
    LAST);
```

In the following example, since all the relevant links have the same class name, newmerc-left-ct, you can perform a random click using the following code:

```
web_text_link("Click",
                DESCRIPTION,
                "Class=newmerc-left-ct",
                "Ordinal=random",
                LAST);
```

The following functions do not support the custom description arguments: **web_browser**, **web_map_area**, **web_radio_group**, and **web_reg_dialog**.

50

Web (HTTP/HTML, Click and Script) Functions

VuGen helps you create Web Vuser scripts that describe user actions on Web sites. Each script contains functions that correspond directly to each of the actions taken.

This chapter includes:

The following information applies to Web (Click and Script), Web (HTTP/HTML), Oracle Web Applications 11i, and PeopleSoft Enterprise.

About Web Vuser Functions

The functions developed to emulate Internet communication between a browser and a Web server are called Web Vuser functions. Each Web Vuser function has a **web** prefix. Some functions are generated when you record a script; others you must manually insert into the script.

The Web functions are categorized as follows:

- Action Functions
- Authentication Functions
- Check Functions
- Connection Definition Functions
- Concurrent Group Functions
- Cookie Functions
- Correlation Functions
- Filter Functions
- Header Functions
- Proxy Server Functions
- Replay Functions
- Miscellaneous Functions

Web (Click and Script) Vusers use other functions to emulate user actions.

Most functions which are not Action functions, may be used in Web (Click and Script) Vuser scripts. However, the **web_concurrent_start** and **web_concurrent_end** functions are specific to Web (HTTP/HTML) Vuser scripts.

For detailed information and examples of the Web Protocol functions, see the *Online Function Reference* (**Help > Function Reference**).

For more information on adding general Vuser functions to scripts, see Chapter 6, "Enhancing Vuser Scripts."

Adding and Editing Functions

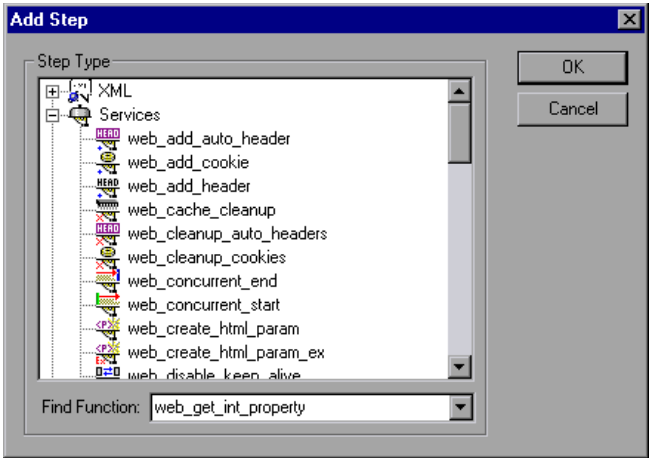
Many of the Web Vuser functions are created during the recording session. You can also manually add and edit Web Vuser functions after recording in both the Tree view and Script view.

When you select a new step to add to your script, VuGen categorizes the steps in the following types:

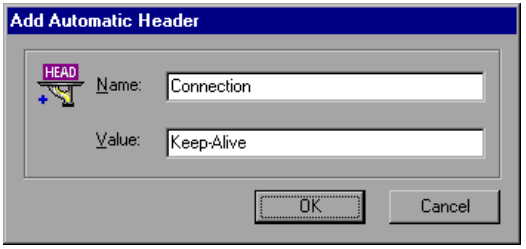
Step Type	Description
Service	A **Service** step is a function that does not make any changes in the Web application context. Rather, service steps perform customization tasks such as setting proxies, providing authorization information, and issuing customized headers.
URL	A **URL** step is generated when you type a URL into the browser or use a bookmark to access a specific Web page. Each URL icon represents a **web_url** function in the Vuser script. The default label of a URL icon is the last part of the URL of the target page.
Link	VuGen adds a **Link** step when you click a hypertext link while recording. Each Link step represents a **web_link** function in the Vuser script. The default label of the step is the text string of the hypertext link (only recorded for the HTML-based recording level).
Image	VuGen adds an **Image** step to the Vuser script when you click a hypergraphic link during recording. Each Image step represents a **web_image** function in the Vuser script. If the image in the HTML code has an ALT attribute, then this attribute is used as the default label of the icon. If the image in the HTML code does not have an ALT attribute, then the last part of the SRC attribute is used as the icon's label (only recorded for the HTML-based recording level).
Submit Form / Submit Data	VuGen adds a **Submit Form** or **Submit Data** step when you submit a form while recording. The default label of the step is the name of the executable program used to process the form (Submit Form only recorded for the HTML-based recording level).
Custom Request	VuGen adds a **Custom Request** step to a Vuser script when you record an action that VuGen cannot recognize as any of the standard actions (i.e., URL, link, image, or form submission). This is applicable to non-standard HTTP applications.

To add a new function to an existing Vuser script:

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



2 Select the desired function and click **OK**. Most Web Vuser functions are under the **Services** category. The Properties dialog box for that function opens. This dialog box lets you specify the function's arguments.



3 Specify the properties and click **OK**. VuGen inserts the function with its arguments at the location of the cursor.

You can edit existing steps by opening the Properties dialog box and modifying the argument values. This is only valid for protocols that support tree view (not available for WAP).

To edit an existing step:

1. In the tree view, select **Properties** from the right-click menu. The Properties dialog box for that function opens.
2. Modify the argument values as necessary and click **OK**.

You can manually add general Vuser functions such as transactions, rendezvous, comments, and log functions during recording. For more information, see Chapter 6, "Enhancing Vuser Scripts."

General Web (Click and Script) API Notes

This section lists general notes about the Web (Click and Script) functions. Note that you can specify a regular expression for most object descriptions, by preceding the text with "/RE" before the equals sign. See the Function Reference (Help > Function Reference) for more details. For example:

```
web_text_link("Manage Assets",
        DESCRIPTION,
        "Text/RE=(Manage Assets)|(Configure Assets)",
        ACTION,
        "UserAction=Click",
        LAST);
```

Ordinals

The Ordinal attribute is a one-based index to distinguish between multiple occurrences of objects with identical descriptions. In the following example, the two recorded **web_text_link** functions have identical arguments, except for the ordinal. The ordinal value of 2, indicates the second occurrence.

```
web_text_link("Manage Assets",
        DESCRIPTION,
        "Text=Manage Assets",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);

web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Ordinal=2",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

Empty Strings

There is a difference between not specifying an argument and specifying it as an empty string. When you do not specify an argument, VuGen uses the default value or ignores it. When you list an argument, but assign it an empty string as a value, VuGen attempts to find a match with an empty string or no string at all. For example, omitting the id argument instructs VuGen to ignore the id property of the HTML element. Specifying "ID=" searches for HTML elements with no id property or with an empty ID.

```
web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Id=",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

Using Cache Data

You can save stored data into your browser's cache, and load it at a later point in the script.

To implement this within your script, you manually add the **web_dump_cache** and **web_load_cache** functions.

Dumping Information to the Cache

Transfering data to the cache is called dumping the information. You run the **web_dump_cache** function to create a cache file in the location specified in the **FileName** argument. You only need to run this function once to generate the cache file.

In the following example, the web_dump_cache function creates a cache file in C:\temp for each VuserName parameter running the script.

```
web_dump_cache("paycheckcache","FileName=c:\\temp\\{Vuser
    Name}paycheck", "Replace=yes", LAST)
```

If you run a single Vuser user ten times, VuGen creates ten cache files in the following format, where the prefix is the VuserName value:

```
Ku001paycheck.cache
Ku002paycheck.cache
Ku003paycheck.cache
…
```

You can modify the first and second arguments (paycheckcache and paycheck in this example) to reflect the current transaction name. Place this function at the end of your script, after you have loaded all of the resources.

Loading Information from the Cache

The **web_load_cache** function loads a cache file whose location is specified in the **FileName** argument. Note that the **web_load_cache** function requires the cache file to exist. Therefore, you can only run this function after running **web_dump_cache**.

In the following example, the **web_load_cache** function loads the **paycheck** cache files from **C:\temp**.

```
web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",LAST)
```

Inserting the Caching Functions

The following procedur describes how to use caching functions.

To use the caching functions:

1. Insert the **web_dump_cache** function into your script.
2. Run the script at least once.
3. Insert the **web_load_cache** function into your script before the Vuser actions.
4. Comment out the **web_dump_cache** function.
5. Run and save the script.

Caching Example

The following example illustrates a PeopleSoft Enterprise Vuser viewing the details of his paycheck.

```
Action()
{
//    web_add_cookie("storedCookieCheck=true; domain=pbntas05; path=/");

web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",LAST);

    web_browser("signon.html",
        DESCRIPTION,
        ACTION,
        "Navigate=http://pbntas05:8200/ps/signon.html",
        LAST);
    lr_think_time(35);

    web_edit_field("userid",
        "Snapshot=t1.inf",
        DESCRIPTION,
        "Type=text",
        "Name=userid",
        ACTION,
        "SetValue={VuserName}",
        LAST);
```

```
    web_edit_field("pwd",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=password",
        "Name=pwd",
        ACTION,
        "SetValue=HCRUSA_KU0007",
        LAST);

    lr_start_transaction("login");
        web_button("Sign In",
            "Snapshot=t3.inf",
            DESCRIPTION,
            "Type=submit",
            "Tag=INPUT",
            "Value=Sign In",
            LAST);
    lr_end_transaction("login", LR_AUTO);

    web_image_link("CO_EMPLOYEE_SELF_SERVICE",
        "Snapshot=t4.inf",
        DESCRIPTION,
        "Alt=",
        "Name=CO_EMPLOYEE_SELF_SERVICE",
        "Ordinal=1",
        ACTION,
        "ClickCoordinate=10,10",
        LAST); …

    web_text_link("Sign out",
        "Snapshot=t7.inf",
        DESCRIPTION,
        "Text=Sign out",
        "FrameName=UniversalHeader",
        ACTION,
        "UserAction=Click",
        LAST);

/*web_dump_cache("paycheck","FileName=c:\\{VuserName}paycheck",
"Replace=yes", LAST);*/
    return 0;
}
```

51

Web (HTTP/HTML, Click and Script) Text and Image Verification

You can add checks to your Web Vuser scripts to determine whether or not the correct Web pages are returned by the server when you run the Vuser script.

This chapter includes:

About Verification Under Load

VuGen enables you to add checks to your Web Vuser scripts. A Web check verifies the presence of a specific object on a Web page. The object can be a text string or an image.

Web checks enable you to determine whether or not your Web site is functioning correctly while it is being accessed by many Vusers—that is, does the server return the correct Web pages? This is particularly important while your site is under the load of many users, when the server is more likely to return incorrect pages.

For example, assume that your Web site displays information on the temperatures in major cities around the world. You use VuGen to create a Vuser script that accesses your Web site.

The Vuser accesses the site and executes a text check on this Web page. For example, if the word **Temperature** appears on the page, the check passes. If **Temperature** does not appear because, for example, the correct page was not returned by the server, the check fails. Note that the text check step appears before the URL step. This is because VuGen registers, or prepares in advance, the search information relevant for the next step. When you run the Vuser script, VuGen conducts the check on the Web page that follows.

Although the server may display the correct page when you record the script and when a single Vuser executes the script, it is possible that the correct page will not be returned when the server is under the load of many Vusers. The server may be overloaded and may therefore return meaningless or incorrect HTML code. Alternatively, in some instances when a server is overloaded, the server may return a **500 Server Error** page. In both of these cases, you can insert a check to determine whether or not the correct page is returned by the server.

Note: Web checks increase the work of a Vuser, and therefore you may need to run fewer Vusers per load generator. You should use Web checks only where experience has shown that the server sometimes returns an incorrect page.

You can define Web checks during or after recording a Vuser script.

VuGen uses several different Web check icons, each one representing a different check type:

Web Check Icon	Description
Text 	A text check, searching for a specific string in the next action function (**web_reg_find**) or in the entire business process (**web_global_verification**) step.
Text 	A text check, searching for a specific string in the downloaded HTML page using the **web_find** step. For more information, see "Understanding Text Check Functions" on page 814.
Image 	An image check, searching for a specific image on a Web page. For more information, see "Understanding Text Check Functions" on page 814.

This chapter describes how to use VuGen to add Web checks in the tree view. For information about adding checks to the script in the text-based script view, see the *Online Function Reference* (**Help > Function Reference**).

Adding a Text Check

VuGen allows you to add a check that searches for a text string on a Web page. You can add the text check either during or after recording.

When you create a text check, you define the name of the check, the scope of the check, the text you want to check for, and the search conditions.

To add a text check during recording:

1 In the application or Web browser window, select the desired text.

2 Click the **Insert Text check** button on the recording toolbar. VuGen adds a **web_reg_find** function to the script.

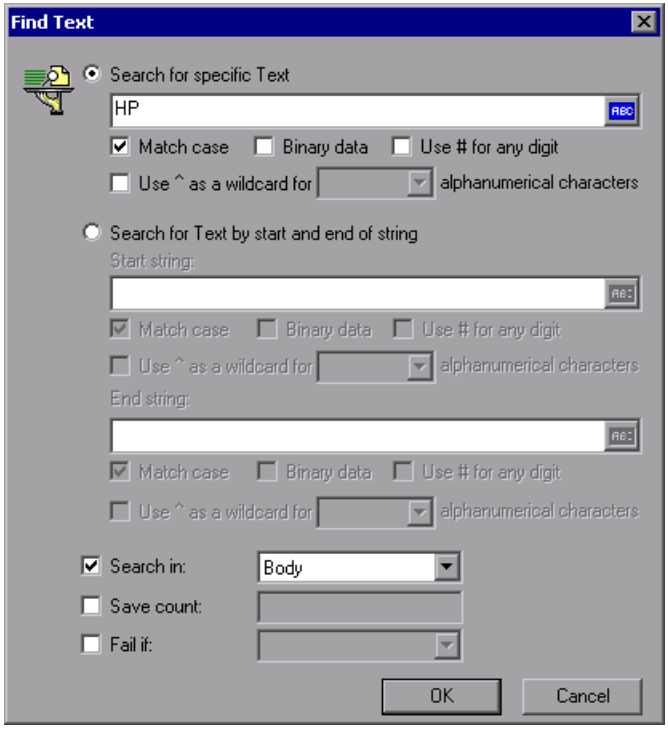
To add a text check after recording:

1 Go to the snapshot of the step whose text you want to check.

2 In the snapshot, select the text you want to verify.

3 Select **Add a Text Check (web_reg_find)** from the right-click menu. The Find Text properties dialog box opens.

Note: For certain protocols, VuGen issues a message indicating that you should add text checks from the Server Response tab—not from the snapshot. Click the **Server Response** tab and select the **HTML Document** tab. Expand the body node and then continue as described below.



The following attributes are available for **web_reg_find**:

- **Search for specific Text.** The text string to search for. This attribute must be a non-empty, null-terminated character string.
- **Search for Text by start and end of string**
  - **Start string.** The prefix of the text string for which you are searching.
  - **End string.** The suffix of the text string for which you are searching.

After specifying a search method—specific text or a beginning or end string, you can specify the search options:

- To perform a case -sensitive search, select **Match case**. To indicate binary data, select **Binary data**. To indicate any digit as a match, use a hash (#) in the text string and select **Use # for any digit**.
- **Use ^ as a wildcard for all/lowercase/uppercase alphanumerical characters.** Allows a wildcard search for alphanumerical characters—either all, uppercase, or lowercase characters. You specify a wildcard with the ^ character.
- **Search in.** Where to search for the text. The available values are **Headers**, **Body**, or **All**. The default is **Body**.
- **Save count.** The number of matches that were found. To use this attribute, select **Save count** and specify a parameter name in which to store the number of matches. The variable will be a null-terminated ASCII value.
- **Fail if.** The handling method when the string is not found. The available values are **Found** and **Not Found**. **Found** indicates that a failure occurs when the text is found (for example, "Error"). **Not Found** indicates that a failure occurs when the text is not found.

To view or modify the properties of the text check after it has been created, click the **Tree View** tab and double-click on the **Services: Reg Find** step. In the Find Text dialog box, you can view or modify all of the step's attributes.

Understanding Text Check Functions

When you add a text check, VuGen adds a **web_reg_find** function to your script. This function registers a search for a text string on an HTML page. Registration means that it does not execute the search immediately—it performs the check only after executing the next Action function, such as **web_url**. Note that if you are working with a concurrent functions group, the **web_reg_find** function is only executed at the end of the grouping.

In the following example, **web_reg_find** function searches for the text string "Welcome". If the string is not found, the next action function fails and the script execution stops.

```
web_reg_find("Text=Welcome", "Fail=Found", LAST);
web_url("Step", "URL=...", LAST);
```

In addition to the **web_reg_find** function, you can use other functions to search for text within an HTML page:

Several additional functions can be used for searching for text:

- web_find
- web_global_verification

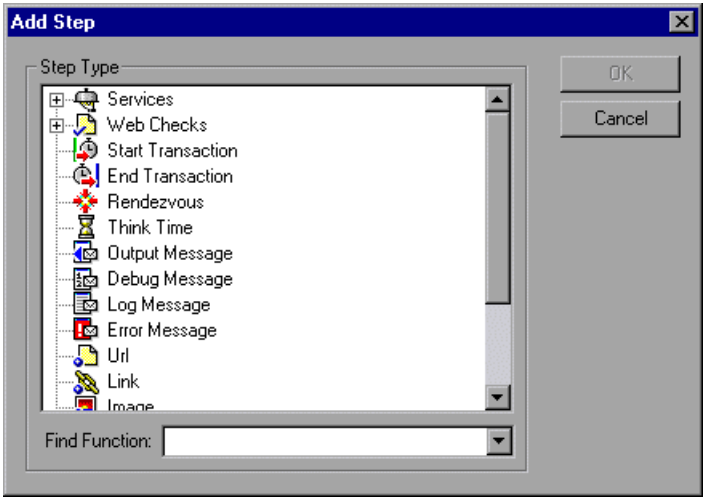
The **web_find** function, primarily used for backward compatibility, differs from the **web_reg_find** function in that **web_find** is limited to an HTML-based script (see **Recording Options** > **Recording** tab). It also has less attributes such as instance, allowing you to determine the number of times the text appeared. When performing a standard text search, **web_reg_find** is the preferred function.

The **web_global_verification** function allows you to search the data of an entire business process. In contrast to **web_reg_find**, which only applies to the next Action function, this function applies to **all** subsequent Action functions such **web_url**. By default, the scope of the search is NORESOURCE, searching only the HTML body, excluding headers and resources.

The **web_global_verification** function is ideal for detecting application level errors that are not included the HTTP status codes. This function is not limited to an HTML-Based script (see **Recording Options** > **Recording** tab).

To add additional functions to your script:

1 In the VuGen main window, click at the point where you want to add the text check. Select **Insert** > **New Step**.



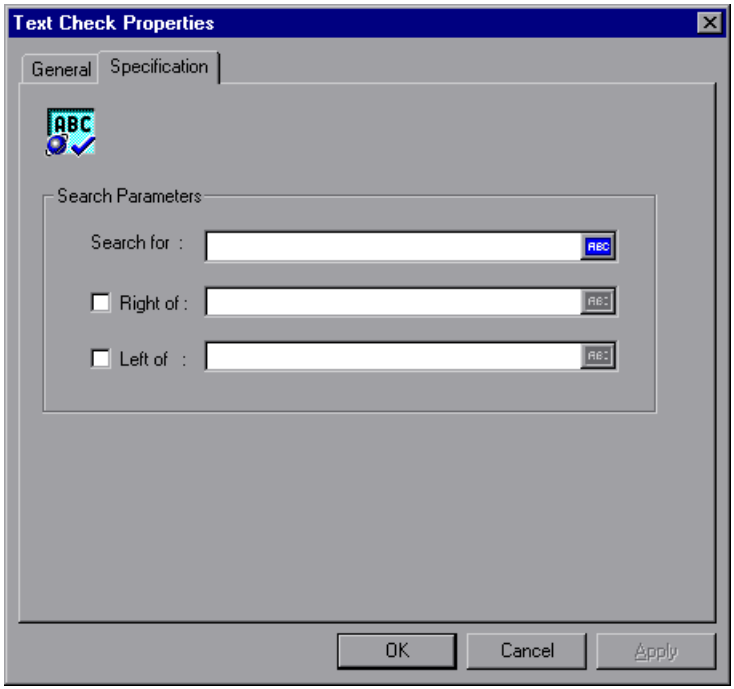
2 For the **web_find** functions, expand the **Web Checks** node and select **Text Check**. For the **web_global_verification** function, expand the **Services** node and select the function name. The Properties dialog box opens.

3 Set the properties for these functions (see description below).

4 Click **OK**. VuGen inserts a new function into the script.

Setting web_find Properties

You can set the following properties for the **web_find** function:

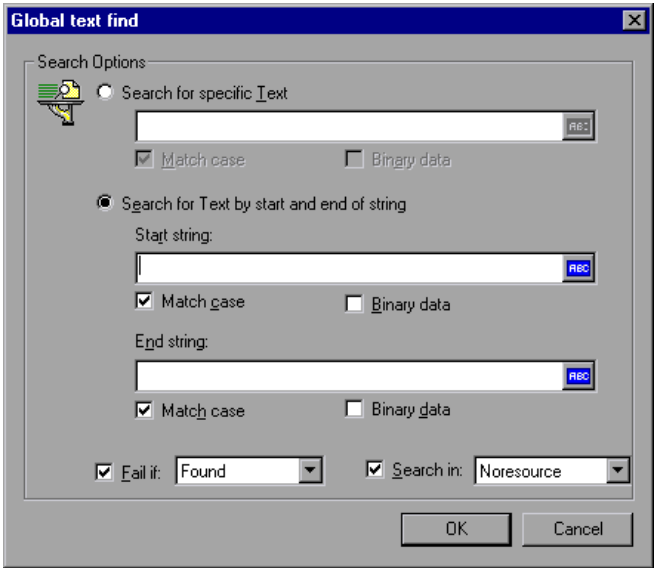


- **Search for**. The string you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."
- **Right of** / **Left of.** The position of the search string relative to adjacent text. Type the text in the appropriate field. For example, to verify that the string support@hp.com appears to the right of the word e-mail:, select **Right of** and then type e-mail: in the **Right of** box.
- **Step Name.** The name of the text check. Click the **General** tab and type a name for the text check in the box. Use a name that you can recognize and identify later on.

Note: A Vuser conducts Web checks during script execution only if checks are enabled, and if the script runs in HTML mode. To enable checks, select the **Enable image and text check** option in the **Preferences** tab in the Run-Time Settings dialog box. For details, see Chapter 79, "Configuring Run-Time Settings."

Setting web_global_verification Properties

You can set the following properties for the **web_find** function:



- **Search for specific text.** The string whose presence you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."
- **Search for Text by Start and End of String.** The boundaries, also known as **Start** and **End** strings that surround the text. Select the appropriate options to indicate if you want to **Match case** or if you are searching for **binary data**.

➤ **Fail if.** Fails the script if the condition is met. You can also indicate the failure condition: if the text is **Found** or **Not found**. Select the desired behavior in the **Fail if** box.

Text Flags

When specifying search text using a registered search, **web_reg_find**, you can add flags to control the scope of the search:

/IC to ignore the case.

/BIN to specify binary data.

/DIG to interpret the pound sign (#) as a wildcard for a single digit. The DIG flag does not match a literal pound sign.

/ALNUM<case> to interpret the caret sign (^) as a wildcard for a single US-ASCII alphanumeric character. There are three syntaxes: **ALNUMIC** to ignore case, **ALNUMLC** to match only lower case, and **ALNUMUC** to match only upper case. The ALNUM flag does not match a literal caret.

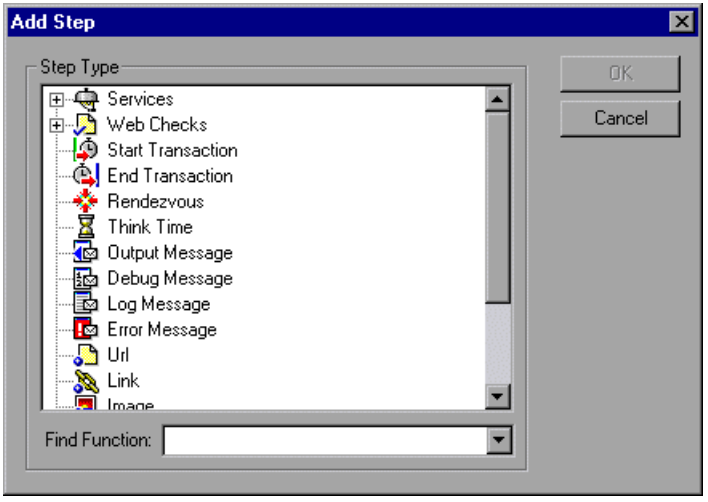
To use flags, you enter the attribute **TEXT**, followed by a forward slash and the flag name. For example, to search for a string ignoring the case, use "Text/IC=search_text"'.

Adding an Image Check

VuGen allows you to add a user-defined check that searches for an image on a Web page. The image can be identified by the ALT attribute, the SRC attribute, or both.

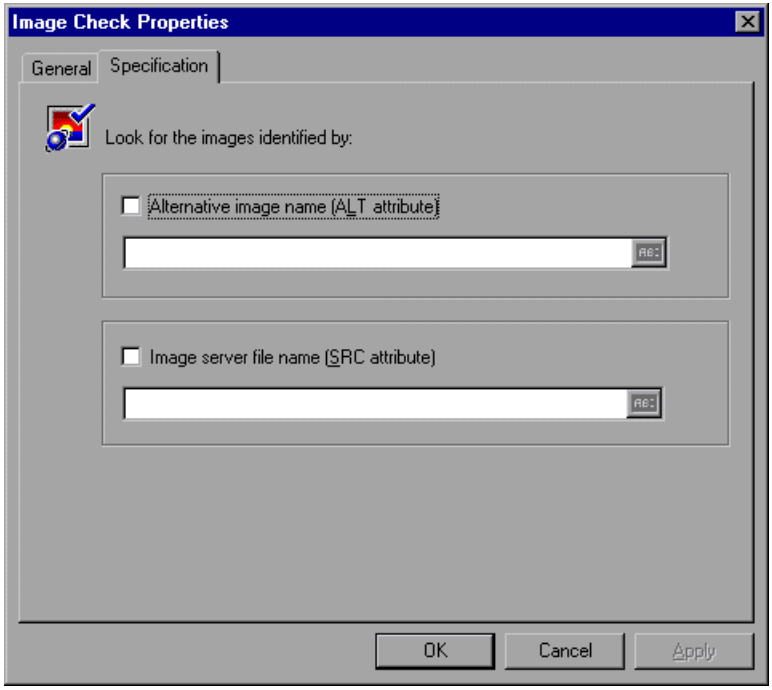
To add an image check:

1 In the VuGen main window, right-click the step corresponding to the Web page on which you want to perform a check. Select **Insert After** from the pop-up menu. The Add Step dialog box opens.



2 Expand **Web Checks** in the **Step Type** tree.

3 Select **Image Check**, and click **OK**. The Image Check Properties dialog box opens. Make sure that the **Specification** tab is visible.



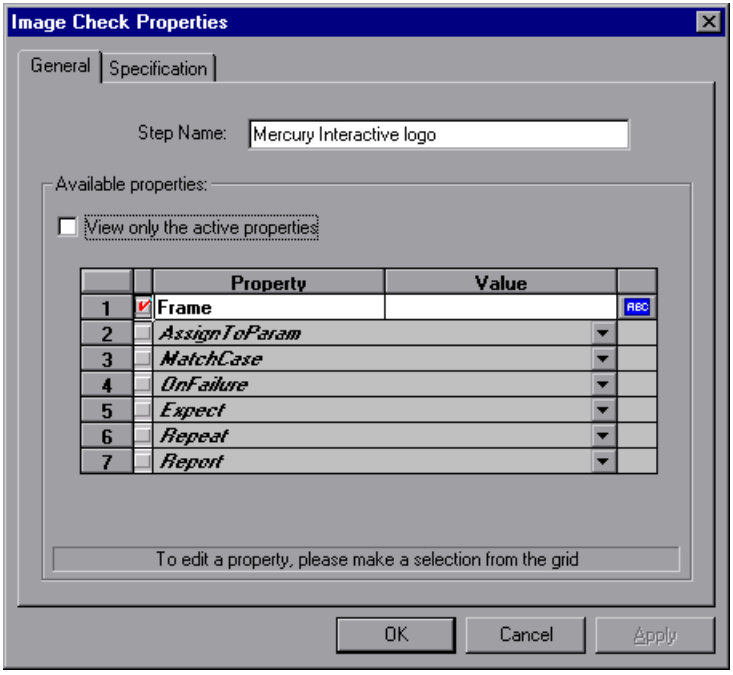
4 Select a method to identify the image:

- **Alternative image name (ALT attribute).** Identifies the image using its **ALT** attribute. Type the ALT attribute text in the text box. When you run the script, the Vuser searches for an image that has the specified ALT attribute.
- **Image server file name (SRC attribute).** Identifies the image using the SRC attribute. Type the SRC attribute text into the text box. When you run the script, the Vuser searches for an image that has the specified SRC attribute.

An ABC icon indicates that the ALT or SRC attribute has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

Note: If you select both the ALT attribute and SRC attribute check boxes, the Vuser searches for an image that has both the specified ALT attribute and the specified SRC attribute.

5 To name the image check, click the **General** tab. In the **Step Name** box, type a name for the image check. Use a name that you can recognize later on.

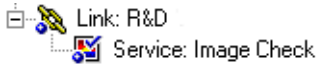


6 The properties table displays additional properties that define the check.

Clear the **View only the active properties** check box to view active and non- active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column.

For details on assigning property values, see "Defining Additional Properties" on page 823.

7 Click **OK** to accept the settings. An icon representing the new image check is added to the associated step in the Vuser script.



Defining Additional Properties

You can specify additional properties for each Web check that you insert into a Vuser script. You set additional options in the properties table on the **General** tab of the check properties dialog boxes. The following is only relevant for **web_find** and **web_image_check** functions—not **web_reg_find**.

To set additional properties:

1 Right-click the Web check whose properties you want to edit, and select **Properties** from the pop-up menu. The appropriate check properties dialog box opens. Make sure that the **General** tab is visible.

2 Clear the **View only the active properties** check box to view all the available properties.

3 To enable a property, click the cell to the left of the property name. A red check mark appears beside the property.

4 Assign the property a value in the **Value** column:

- **Frame.** Type the name of the frame where the check object is located.
- **AssignToParam.** Select **YES** to enable assigning to a parameter. Select **NO** to disable this capability. The default value is **NO**.
- **MatchCase.** Select **YES** to conduct a case-sensitive search. Select **NO** to conduct a non case-sensitive search. The default value is **NO**.

- **OnFailure.** Select **Abort** to abort the entire Vuser script if the check fails. VuGen aborts the Vuser script regardless of the error-handling method that has been set in the run-time settings. Select **Continue** to have the error-handling method defined in the run-time settings determine whether or not the script is aborted if the check fails.

  The default value is **Continue**. For details on defining the error handling method, see Chapter 79, "Configuring Run-Time Settings."

- **Expect.** Select **NotFound** to indicate that the check is successful if the Vuser does not find the specified check object. Select **Found** to indicate that the check is successful if the Vuser finds the specified check object. The default value is Found.

- **Repeat.** Select **YES** to search for multiple occurrences of the specified check object. Select **NO** to end the check as soon as one occurrence of the specified check object is found. The Vuser script continues with the next step. This option is useful when searching through a Web page that may have multiple occurrences of the check object. The default value is YES.

- **Report.** Select **Always** to always view a detailed description of the check results in the Execution Log. Select **Failure** to view detailed check results only when the check fails. Select **Success** to view detailed check results only when the check succeeds. The default value is **Always**.

An ABC icon indicates that the property value has not been assigned a parameter. Click the icon to assign a parameter. For more information, see Chapter 70, "Working with VuGen Parameters."

52

Modifying Web and Wireless Vuser Scripts

After recording a Web or Wireless Vuser script, you use VuGen to modify the recorded script. You can add new steps, and edit or delete existing steps.

This chapter includes:

About Modifying Web and Wireless Vuser Scripts

After recording a session, you can modify the recorded script in VuGen by editing a step's properties or adding and deleting steps.

You can do the modifications either in the icon-based tree view or in the text-based script view. For details on the two viewing modes, see Chapter 48, "Web (HTTP/HTML, Click and Script) Protocols."

This chapter describes how to use VuGen to modify the script in the tree view. For information about modifying the script in the text-based script view, see the *Online Function Reference* (**Help > Function Reference**).

Adding Binary Data

To include binary coded data in the body of an HTTP request, use the following format:

\x[char1][char2]

This represents the hexadecimal value that is represented by [char1][char2].

For example, \x24 is 16*2+4=36, is a $ sign, and \x2B is a + sign.

Do not use single-character hexadecimal sequences. For example, \x2 is not a valid sequence but \x02 is.

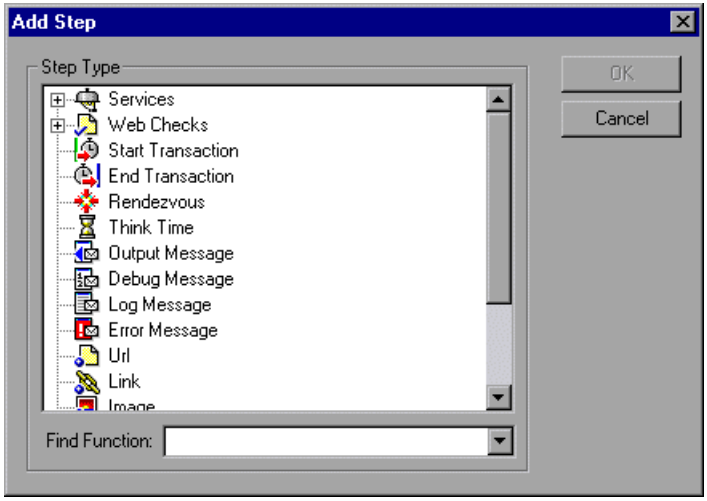
Adding a Step to a Vuser Script

In addition to the steps that VuGen records during the recording session, you can add steps to a recorded script.

To add a step to a Vuser script:

1 In the tree view of the script, select the step before or after which you want to add the new step.

2 Select **Insert** > **New Step** to insert a step after the selected step, or select **Insert After** or **Insert Before from** the right-click menu. The Add Step dialog box opens.



3 Select the type of step you want to add from the **Step Type** tree or from the **Find Function** list.

4 Click **OK**. An additional dialog box opens, prompting for information about the step to add. This dialog box varies, depending on the type of step that you are adding.

For details on using these dialog boxes, see the appropriate section, as listed below:

To add this...	See...
Vuser API function	Chapter 6, "Enhancing Vuser Scripts"
Service step	"Modifying Service Steps" on page 849
Web Check	"Modifying Web Checks (Web only)" on page 850
Transaction	"Modifying a Transaction" on page 846
Rendezvous point	"Modifying a Rendezvous Point" on page 847
Think time step	"Modifying Think Time" on page 848
URL step	"Modifying a URL Step" on page 829
Link step	"Modifying a Hypertext Link Step (Web only)" on page 831
Image step	"Modifying an Image Step (Web only)" on page 833
Submit form step	"Modifying a Submit Form Step (Web only)" on page 835
Submit data step	"Modifying a Submit Data Step" on page 839
Custom request step	"Modifying a Custom Request Step" on page 843
User-defined step	Chapter 6, "Enhancing Vuser Scripts"

Deleting Steps from a Vuser Script

After recording a session, you can use VuGen to delete any step from the Vuser script.

To delete a step from a Vuser script:

1. In the tree view of the Vuser script, right-click the step you want to delete, and select **Delete** from the pop-up menu.
2. Click **OK** to confirm that you want to delete the step.

The step is deleted from the script.

Modifying Action Steps

An action step represents a user action during recording, that is, a jump to a new URL or a change in the Web context.

Action steps, represented in the tree view of the Vuser script by Action icons, are added to your script automatically during recording. After recording, you can modify the recorded action steps.

This section includes:

- Modifying a URL Step
- Modifying a Hypertext Link Step (Web only)
- Modifying an Image Step (Web only)
- Modifying a Submit Form Step (Web only)
- Modifying a Submit Data Step
- Modifying a Custom Request Step

Modifying a URL Step

A URL step is added to the Vuser script when you type in a URL or use a bookmark to access a specific Web page.

The properties that you can modify are the name of the step, the address of the URL, target frame, and record mode.

By default, VuGen runs the URL step, based on the mode in which it was recorded: **HTML**, or **HTTP** (without resources). For information on the recording modes, see "Understanding the Recording Levels" on page 1218.

Setting the Replay Mode

In the URL step's Properties dialog box, you can modify the mode settings to instruct Vusers to execute the script in a mode other than the recorded mode. To customize the replay mode, select the **Record mode** check box. The available replay modes are:

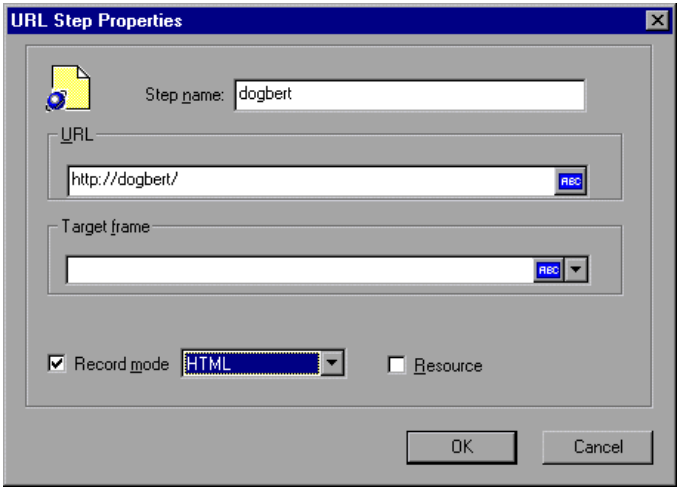
- **HTML.** Automatically download all resources and images and store the required HTTP information for the steps that follow. This is ideal for script with Web links.
- **HTTP.** Do not download any resources for this step during replay. Download only resources that are explicitly represented by functions.

You can also indicate that a certain step should not be counted as a resource. For example, if you have a step that represents a specific image that you want to skip, you can instruct VuGen to exclude that resource type. For more information, see the "Resource Handling" on page 1228.

To modify the properties of a URL step:

1 In the tree view of the Vuser script, select the URL step you want to edit. URL steps are shown using the URL icon.

2 Click the **Properties** button on the VuGen toolbar. The URL Step Properties dialog box opens.



3 To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.

4 In the **URL** box, type the Web address (URL) of the Web page that is accessed by the URL step. An **ABC** icon indicates that the URL has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

5 In the **Target frame** list, select one of the following values:

- **SELF.** Replaces the last (changed) frame.
- **PARENT.** Replaces the parent of the last (changed) frame.
- **TOP.** Replaces the whole page.
- **BLANK.** Opens a new window.

6 To customize the replay mode, select the **Record mode** check box.

Select the desired mode: HTML or HTTP.

7 To exclude an item from being downloaded as a resource, clear the **Resource** check box.

8 Click **OK** to close the URL Step Properties dialog box.

Modifying a Hypertext Link Step (Web only)

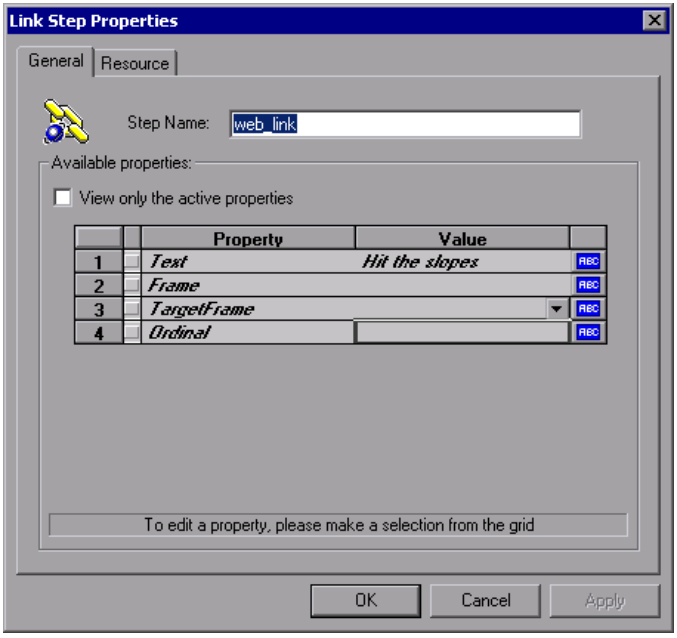
A hypertext link step is added to the Web Vuser script when you click a hypertext link. This step is only recorded when you select the option to record in **HTML based script** mode. See Chapter 77, "Setting Recording Options for Web Vusers." for more information.

The properties that you can modify are the name of the step, how the hypertext link is identified, and where it is located.

To modify the properties of a hypertext link step:

1 In the tree view of the Vuser script, select the hypertext link step you want to edit. Hypertext link steps are shown using the **Hypertext Link** icon.

2 Select **Properties** from the right-click menu. The Link Step Properties dialog box opens.



3 To change the step name, type a new name in the **Step Name** box. The default name during recording is the text string of the hypertext link.

4 The properties table displays the properties that identify the link.

Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **Text.** The exact string of the hypertext link.
- **Frame.** The name of the frame where the link is located.

- **TargetFrame.** The target frame:
  - **TOP.** Replaces the whole page.
  - **BLANK.** Opens a new window.
  - **PARENT**. Replaces the parent of the last (changed) frame.
  - **SELF.** Replaces the last (changed) frame.
- **Ordinal.** a number that uniquely identifies the link when all the other property attributes are identical to one or more other links on the Web page. See the *Online Function Reference* for details.

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

5 Click **OK** to close the Link Step Properties dialog box.

Modifying an Image Step (Web only)

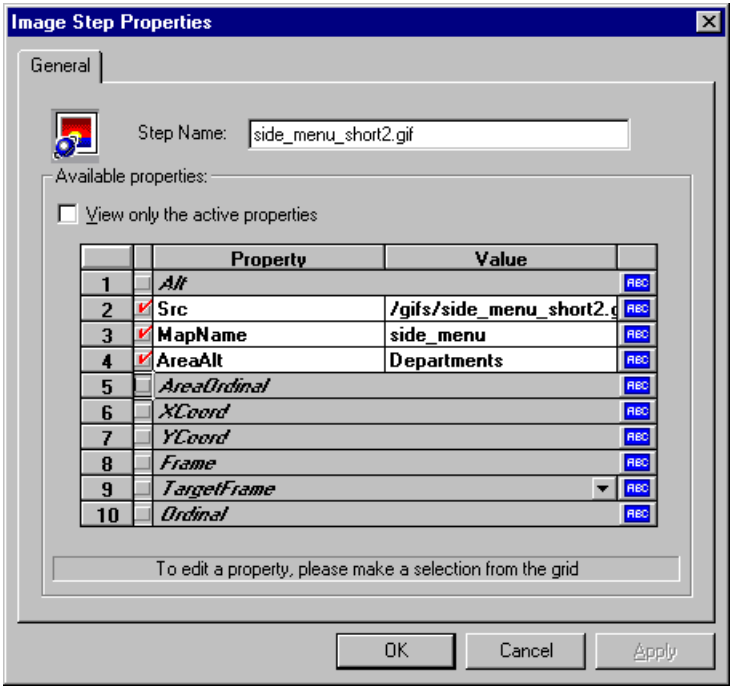
An image step is added to the Vuser script when you click a hypergraphic link. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. See Chapter 77, "Setting Recording Options for Web Vusers." for more information.

The properties that you can modify are the name of the step, how the hypergraphic link is identified, and where it is located.

To modify the properties of an image step:

1 In the tree view of the Vuser script, select the image step you want to edit. Image steps are shown using the **Image** icon.

2 Select **Properties** from the right-click menu. The Image Step Properties dialog box opens.



3 To change the step name, type a new name in the **Step Name** box. The default name during recording is the image's ALT attribute. If the image does not have an ALT attribute, then the last part of the SRC attribute is used as the default name.

4 The properties table displays the properties that identify the link.

Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

➤ **ALT.** The ALT attribute of the image.

- **SRC.** The SRC attribute of the image.
- **MapName.** The name of the map related to the image. Applies to client-side image maps only.
- **AreaAlt.** The ALT attribute of the area to click. Applies to client-side image maps only.
- **AreaOrdinal.** The serial number of the area to click. Applies to client-side image maps only.
- **Frame.** The name of the frame where the image is located.
- **TargetFrame.**The target frame:
  - **_TOP.** Replaces the whole page.
  - **_BLANK.** Opens a new window.
  - **_PARENT**. Replaces the parent of the last (changed) frame.
  - **_SELF.** Replaces the last (changed) frame.
- **Ordinal**. a number that uniquely identifies the image when all other property attributes are identical to one or more other images on the Web page. See the *Online Function Reference* for details.
- **XCoord, YCoord.** The coordinates of the mouse-click on the image.

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

5 Click **OK** to close the Image Step Properties dialog box.

Modifying a Submit Form Step (Web only)

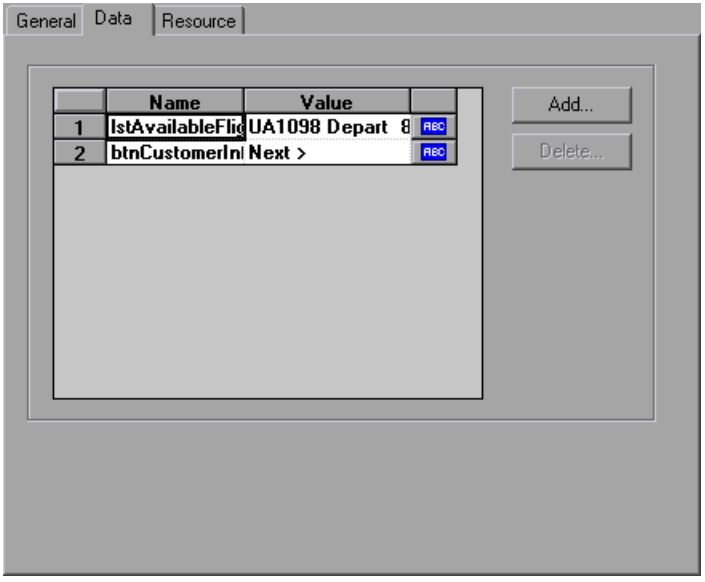
A submit form step is added to the Vuser script when you submit a form. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. See Chapter 77, "Setting Recording Options for Web Vusers." for more information.

The properties that you can modify are the name of the step, the form location, how the form submission is identified, the form data, and the resources for the step.

To modify the properties of a submit form step:

1 In the tree view of the Vuser script, select the submit form step you want to edit. Submit form steps are shown using the **Submit Form** icon.

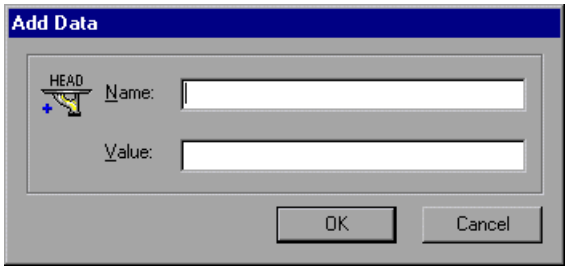
2 Select **Properties** from the right-click menu. The Submit Form Step Properties dialog box opens. Click the **Data** tab.



- The **Name** column lists all the data arguments on the form.
- The **Value** column displays the corresponding value input for a data argument.
- The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 70, "Working with VuGen Parameters," the ABC icon changes to a table icon.

3 To edit a data argument, double-click on it to activate the cursor within the cell and type the new value in the editable box.

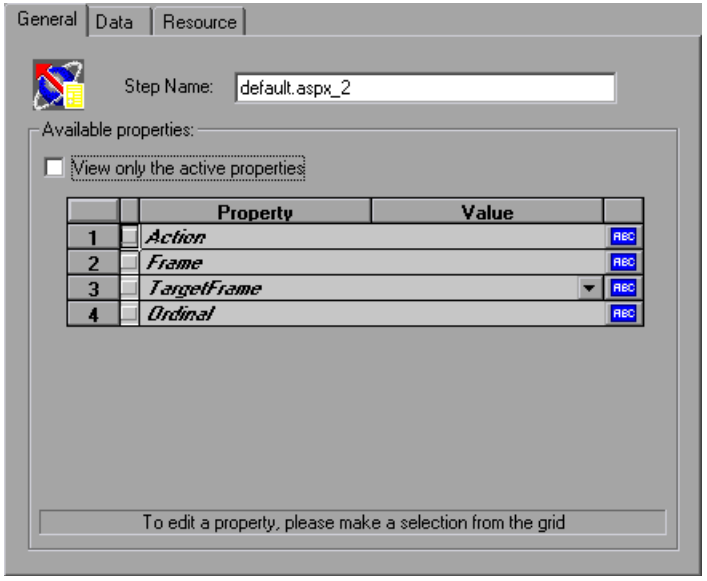
4 To add a new data argument to the form submission, click **Add**. The Add Data dialog box opens.



5 Type a **Name** and **Value** for the data argument, and click **OK**.

6 To delete an argument, select it and click **Delete**.

7 To change the name of the submit form step, click the **General** tab.



8 To change the step name, type a new name in the **Step Name** box. The default name during recording is the name of the executable program used to process the form.

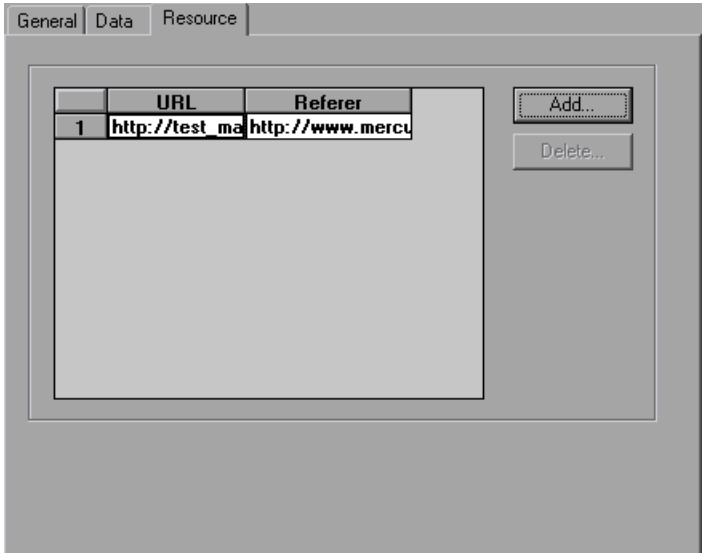
9 The properties table displays the properties that identify the form submission.

Clear the **View only the active properties** option to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **Action.** The address to be used to carry out the action of the form.
- **Frame.** The name of the frame where the form submission is located.
- **TargetFrame.** The target frame:
  - **_TOP.** Replaces the whole page.
  - **_BLANK.** Opens a new window.
  - **_PARENT**. Replaces the parent of the last (changed) frame.
  - **_SELF.** Replaces the last (changed) frame.
- **Ordinal.** a number that uniquely identifies the form when all other property attributes are identical to one or more other forms on the same Web page. See the *Online Function Reference* for details (**Help > Function Reference**).

An ABC icon indicates that the submit form step property value has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

10 To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.



11 Click **OK** to close the Submit Form Step Properties dialog box.

Modifying a Submit Data Step

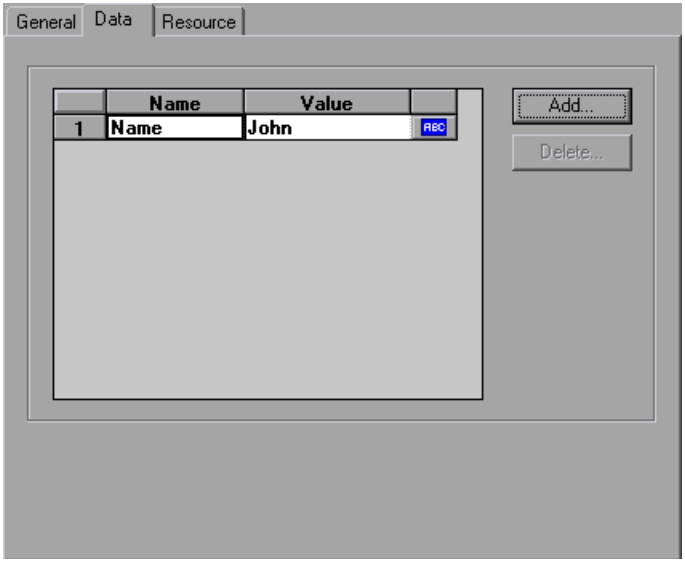
A submit data step represents the submission of data to your Web site for processing. This is different from a Submit Form step because you do not need to have a form context to execute this request.

The properties that you can modify are the name of the step, the method, the action, the target frame, and the data items on the form.

To modify the properties of a submit data step:

1 In the tree view of the Vuser script, select the submit data step you want to edit. Submit data steps are represented by the **Submit Data** icon.

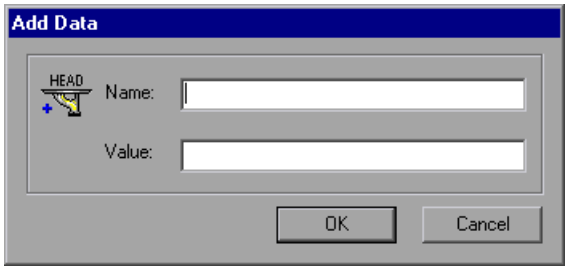
2 Select **Properties** from the right-click menu. The Submit Data Step Properties dialog box opens. Click the **Data** tab.



- The **Name** column lists all the data arguments on the form. This includes all hidden fields.
- The **Value** column displays the corresponding value input for a data argument.
- The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 70, "Working with VuGen Parameters," the **ABC** icon changes to a table icon.

3 To edit a data argument, double-click on it to activate the cursor within the cell. Then type the new value.

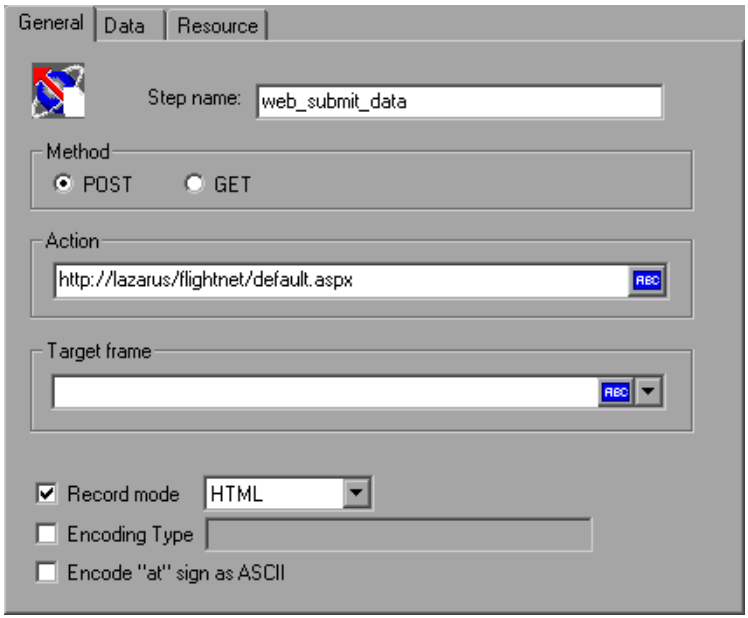
4 To add new data, click **Add**. The Add Data dialog box opens.



5 Type a **Name** and **Value** for the data argument, and click **OK**.

6 To delete an argument, select it and click **Delete**.

7 To change the name of the submit data step, click the **General** tab.



8 To change the step name, type a new name in the **Step name** box.

9 Under **Method**, click **POST** or **GET**. The default method is **POST**.

10 In the **Action** box, type the address to be used to carry out the action of the data submission. An ABC icon indicates that the action has not been assigned a parameter. For details on assigning parameters, see Chapter 70, "Working with VuGen Parameters."

11 Select a **Target frame** from the list:

- **TOP.** Replaces the whole page.
- **BLANK.** Opens a new window.
- **PARENT**. Replaces the parent of the last (changed) frame.
- **SELF.** Replaces the last (changed) frame.

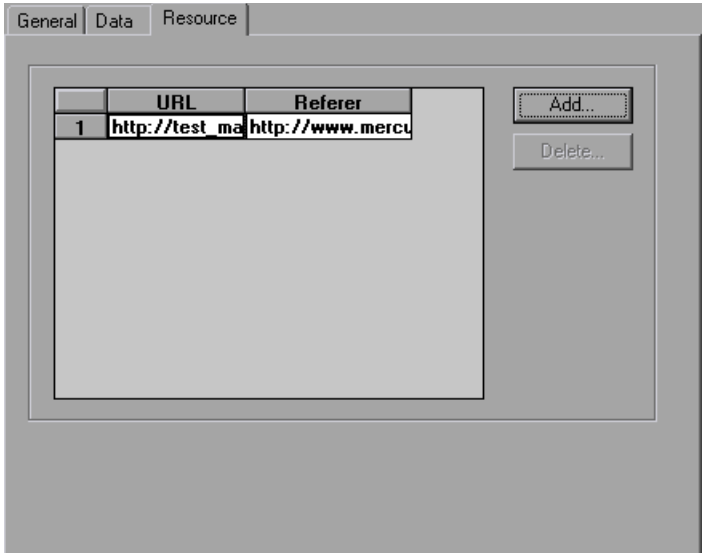
12 To customize the replay mode, select the **Record mode** option. Select the desired mode: HTML, or HTTP. For more information, see "Setting the Replay Mode" on page 830.

13 To specify an encoding type, such as **multipart/www-urlencoded**, select the **Encoding type** check box and specify the encoding method.

14 To encode the "@" in the URL, select **Encode "at" sign as ASCII**.

15 Click **OK** to close the Submit Data Step Properties dialog box.

16 To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.



Modifying a Custom Request Step

A custom request represents a custom HTTP request for a URL, with any method supported by HTTP. A custom request step is contextless.

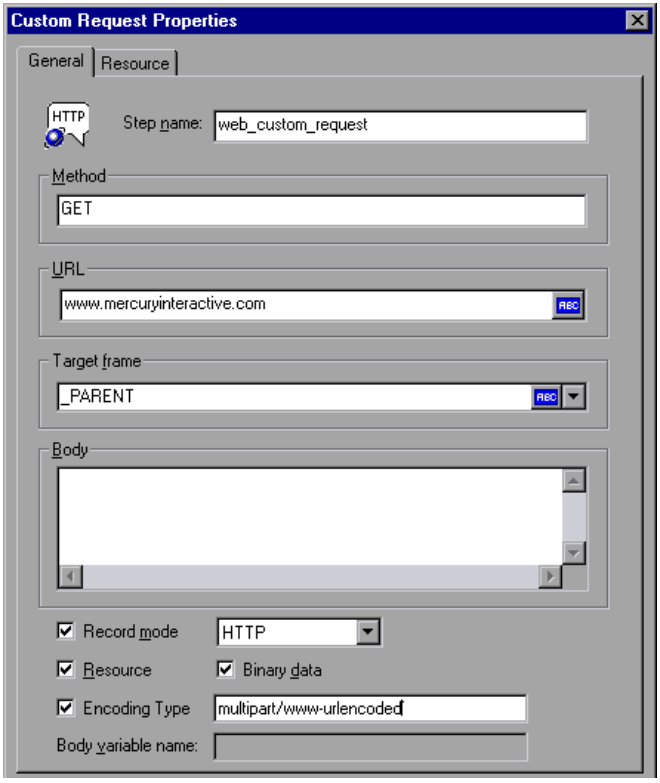
The properties that you can modify are the name of the step, method, URL, target frame, and body.

VuGen has a feature that lets you convert a custom request body string to C format. For example, if you copy an XML tree or a large amount of data into the Body area of the custom request, you can convert the strings to C format in order that it may be incorporated into the current function. VuGen inserts the necessary escape sequence characters and removes the line breaks in the string.

To modify the properties of a custom request step:

1 In the tree view of the Vuser script, select the custom request step you want to edit. Custom request steps are shown using the **Custom Request** icon.

2 Select **Properties** from the right-click menu. The Custom Request Properties dialog box opens.



3 To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.

4 In the **Method** box, type any method supported by HTTP. For example, GET, POST or HEAD.

5 In the **URL** box, type the URL being requested.

6 Select a **Target frame** from the list:

- **TOP.** Replaces the whole page.
- **BLANK.** Opens a new window.
- **PARENT**. Replaces the parent of the last (changed) frame.
- **SELF.** Replaces the last (changed) frame.

7 In the **Body** attribute box, type the body of the request or paste in the desired text. If you select the **Binary data** check box, the text is treated as binary and not as ASCII. For details on using binary data, see the *Online Function Reference* (**Help** > **Function Reference**).

VuGen replaces a Body attribute whose length exceeds 100K, with a variable in the **Body variable name** box. The variable is defined in the **lrw_custom_body.h** file located in the **include** folder.

8 For strings that you pasted into the **Body** box, select the text and select **Convert to C format** from the right-click menu.

9 To customize the replay mode, select the **Record mode** option. Select the desired mode: HTML or HTTP. For more information, see "Setting the Replay Mode" on page 830.

10 To exclude an item from being downloaded as a resource, clear the **Resource** option.

11 To specify an encoding type, such as **multipart/www-urlencoded**, select **Encoding type** and specify the encoding method.

12 Click **OK** to close the Custom Request Properties dialog box.

Modifying Control Steps

A control step represents a non-action step used during load testing. Control steps include transactions, rendezvous points, and think time.

You add control steps, represented in the tree view of the Vuser script by Control icons, to your script during and after recording.

This section includes:

- Modifying a Transaction
- Modifying a Rendezvous Point
- Modifying Think Time

Modifying a Transaction

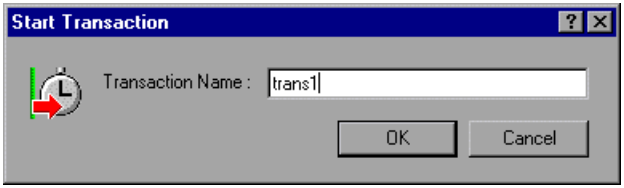
A transaction is a task or set of actions whose server response time you want to measure.

The properties that you can modify are the name of the transaction (start transaction and end transaction) and its status (end transaction only).

To modify a start transaction control step:

1 In the tree view of the Vuser script, select the start transaction control step you want to edit. Start transaction control steps are shown using the **Start Transaction** icon.

2 Select **Properties** from the right-click menu. The Start Transaction dialog box opens.

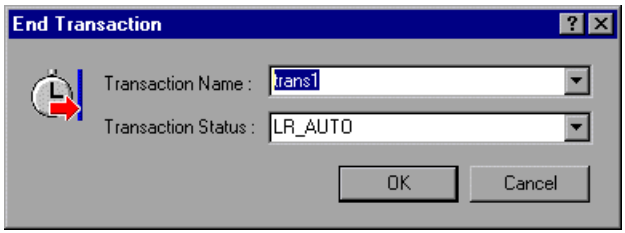


3 To change the transaction name, type a new name in the **Transaction Name** box, and click **OK**.

To modify an end transaction control step:

1 In the tree view of the Vuser script, select the end transaction control step you want to edit. End transaction control steps are shown using the **End Transaction** icon.

2 Select **Properties** from the right-click menu. The End Transaction dialog box opens.



3 Select the name of the transaction you want to end from the **Transaction Name** list.

4 Select a transaction status from the **Transaction Status** list:

- **LR_PASS.** Returns a "succeed" return code.
- **LR_FAIL.** Returns a "fail" return code.
- **LR_STOP.** Returns a "stop" return code.
- **LR_AUTO.** Automatically returns the detected status.

For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

5 Click **OK** to close the End Transaction dialog box.

Modifying a Rendezvous Point

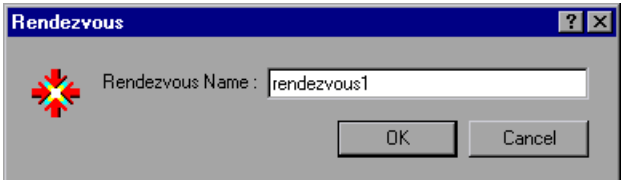
Rendezvous points enable you to synchronize Vusers to perform a task at exactly the same moment.

The property that you can modify is the name of the rendezvous point.

To modify a rendezvous point:

1 In the tree view of the Vuser script, select the rendezvous point you want to edit. Rendezvous points are shown using the **Rendezvous** icon.

2 Select **Properties** from the right-click menu. The Rendezvous dialog box opens.



3 To change the rendezvous name, type a new name in the **Rendezvous Name** box, and click **OK**.

Modifying Think Time

Think time emulates the time that a real user waits between actions. During recording, VuGen automatically adds think time to the Vuser script after each user action—if the time between that action and the subsequent action exceeds the predefined threshold.

The property that you can modify is the think time, in seconds.

To modify the think time:

1 In the tree view of the Vuser script, select the think time step you want to edit. Think time steps are shown using the **Think Time** icon.

2 Select **Properties** from the right-click menu. The Think Time dialog box opens.

3 Type a think time in the **Time To Think** box, and click **OK**.

Note: When you run a Web Vuser script, you can instruct the Vuser to replay think time as recorded or ignore the recorded think time. For details, see Chapter 79, "Configuring Run-Time Settings."

Modifying Service Steps

A service step is a function that performs customization tasks such as setting proxies, submitting authorization information, and issuing customized headers. Service steps do not make any changes to the Web site context.

You add service steps to your script during and after recording.

To modify the properties of a service step:

1. In the tree view of the Vuser script, select the service step you want to edit. Service steps are shown using the Service icon.
2. Select **Properties** from the right-click menu. The appropriate service step properties dialog box opens. This dialog box varies, depending on the type of service step that you are modifying. A description of the service step is displayed in the title bar of the dialog box.

   Note: Some service step functions have no arguments. In these cases, the Properties menu item is disabled.
3. Type or select the arguments required for the service step. See the *Online Function Reference* for details of each function (**Help** > **Function Reference**).
4. Click **OK** to close the service step properties dialog box.

Modifying Web Checks (Web only)

A Web check is a function that verifies the presence of a specific object on a Web page. The object can be a text string or an image.

You add Web checks to your script during and after recording.

To modify the properties of a Web check:

1 In the tree view of the Vuser script, select the Web check you want to edit. Web checks are shown using Web Check icons.

Icon	Description
	Image Check icon
	Text Check icon

2 Select **Properties** from the right-click menu. The appropriate Web check properties dialog box opens. This dialog box varies, depending on the type of check that you are modifying.

3 Type or select the properties required for the check. For details, see Chapter 51, "Web (HTTP/HTML, Click and Script) Text and Image Verification."

4 Click **OK** to close the check properties dialog box.

53

Web (HTTP/HTML) Correlation Rules

VuGen's correlation feature allows you to link Vuser functions by using the results of one statement as input for another.

This chapter includes:

About Correlating Statements

HTML pages often contain dynamic data, which is data that changes each time you access a site. For example, certain Web servers use links comprised of the current date and time.

When you record a Web Vuser script, dynamic data may be recorded into the script. Your script tries to present the recorded values to the Web server, but they are no longer valid. The Web server rejects them and issues an error. These errors are not always obvious, and you may only detect them by carefully examining Vuser log files.

If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

The dynamic data in an HTML page can be in the form of:

- a URL that changes each time you access the associated Web page
- form field (sometimes hidden) value(s).
- JavaScript cookies

Case 1

Suppose a Web page contains a hypertext link with text: "Buy me now!" When you record a script with HTTP data, the URL is recorded by VuGen as:

```
"http://host//cgi-bin/purchase.cgi?date=170308&ID=1234"
```

Since the date "170308" and ID "1234" are created dynamically during recording, each new browser session recreates the date and ID. When you run the script, the link "Buy me now!" is no longer associated with the same URL that was recorded—but with a new one. The Web server is therefore unable to retrieve the URL.

Case 2

Consider a case where a user fills in his name and account ID into a form, and then submits the form.

When the form is submitted, a unique serial number is also sent to the server together with the user's data. Although this serial number is contained in a hidden field in the HTML code, it is recorded by VuGen into the script. Because the serial number changes with each browser session, Vusers were unable to successfully replay the recorded script.

You can use correlated statements to resolve the difficulties in both of the above cases. Replace the dynamic data in the recorded script with one or more parameters. When the script runs, it assigns values to each of the parameters.

Understanding the Correlation Methods

This chapter discusses automatic correlation using built-in or user-defined rules. To manually correlate statements, or to perform correlation for Wireless Vuser scripts, see "Performing Manual Correlation" on page 875.

When recording a browser session, you should first try recording in HTML mode. This mode decreases the need for correlation. For more information about the various recording modes, see "Understanding the Recording Levels" on page 1218.

You can instruct VuGen to correlate the statements in your script either during or after recording. The recording-time solutions described in this chapter automatically correlate the statements in your script during recording time. You can also use VuGen's snapshot correlation to correlate scripts after recording. For more information on correlating after recording, see Chapter 54, "Web (HTTP/HTML) -Correlation After Recording."

Using VuGen's Correlation Rules

VuGen's correlation engine allows you to automatically correlate dynamic data during your recording session using one of the following mechanisms:

- Built-in Correlation
- User-Defined Rule Correlation

For additional information, see "Adding Match Criteria" on page 858 and "Advanced Correlation Rules" on page 858.

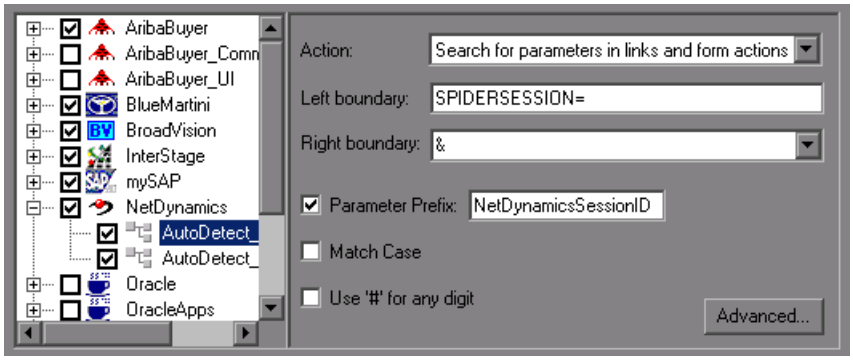
Built-in Correlation

The Built-in correlation detects and correlates dynamic data for supported application servers. Most servers have clear syntax rules, or contexts, that they use when creating links and referrals.

For example, BroadVision servers create session IDs that are always placed between the same delimiters: "BV_SessionID=" on the left, and "&" on the right.

```
BV_SessionID=@@@@1303778278.0969956817@@@@&
```

If you are recording a session with a supported application server, you can use one of the existing rules built into VuGen. An application server may have more than one rule. You can enable or disable a specific rule by selecting or clearing the check box adjacent to the rule. VuGen displays the rule definitions in the right pane.



If you are recording a session on an unsupported application server whose context is not known, and you cannot determine any correlation rules, you can use VuGen's snapshot comparison method. This method guides you through the correlation procedure after you finish recording. For more information, see Chapter 54, "Web (HTTP/HTML) -Correlation After Recording."

User-Defined Rule Correlation

If your application has unique rules and you are able to determine them clearly, you can define new rules using the Recording Options.

User-defined rule correlation requires you to define correlation rules before you record a session. You create the correlation rules in the Recording Options dialog box. The rules include information such as the boundaries of the dynamic data you want to correlate and other specifications about the match such as binary, case matching, and the instance number.

You instruct VuGen where to search for the criteria:

- All Body Text
- Link/Form Actions
- Cookie Headers
- Form Field Value
- Insert Cookie Function

Note that by default, the maximum size of a string that you can save for a rule is 4096 characters. If necessary, you can modify this value by increasing the value of the **MaxParamLen** attribute in the **CorrelationSettings.xml** file, located in the Windows Installation directory.

All Body Text

The **Search for Parameters in all of the Body Text** option instructs the recorder to search the entire body—not just links, form actions or cookies. It searches the text for a match using the borders that you specify.

Link/Form Actions

The **Search for parameters in links and form actions** method instructs VuGen to search within links and forms' actions for the text to parameterize. This method is for application servers where you know the context rules. You define a left boundary, a right boundary, an alternate right boundary, and an instance (occurrence) of the left boundary within the current link.

For example, suppose you want to replace any text between the second occurrence of the string "sessionid=" and "@" with a parameter. Specify **sessionid=** as a left boundary in the **Left Boundary** box, and **@** as a right boundary in the **Right Boundary** box. Since you are looking for the second occurrence, select **second** in the **Instance** box.

If the right boundary is not consistent, you can specify an alternate right boundary in the **Alternate right boundary** box. It uses this value when it cannot uniquely determine the specified right boundary.

For example, suppose the Web page contains links in the following formats:

```
"SessionID=122@page.htm"
"Page.htm@SessionID=122&test.htm"
```

Specifying the right boundary alone is not sufficient, since it is not consistent—sometimes it is "@" and other times it is "&". In this case, you specify "&" as the alternate right boundary.

The left and right boundaries should uniquely identify the string. Do not include dynamic data in the boundaries. You can also specify **End of String** or **Newline Character** as a right boundary, available as options in the drop-down menu.

Note that for this option, the left and right boundaries must appear in the string that appears in the script—it is not sufficient for the boundaries to be returned by the server. This limitation does not apply to the other action types.

Cookie Headers

The **Search for Parameters from Cookie Header** method is similar to the previous rule, except that the value is extracted from cookie text (exactly as it appears in the recording log) instead of from a link or form action.

Form Field Value

The **Parameterize form field value** method instructs VuGen to save the named form field value to a parameter. It creates a parameter and places it in the script before the form's action step. For this option, you need to specify the field name.

Insert Cookie Function

The **Text to enter a web_reg_add_cookie function by** method inserts a **web_reg_add_cookie** function if it detects a certain string in the buffer. It only adds the function for those cookies with the specified prefix. For this option, you need to specify the search text and the cookie prefix.

Adding Match Criteria

In addition to the above rules, you can further define the type of match for your correlation by specifying the following items for the string:

- **Parameter Prefix.** Uses a prefix in all automatically generated parameters based on this rule. Prefixes prevent you from overwriting existing user parameters. In addition, prefixes allow you to recognize the parameter in your script. For example, in Siebel Web, one of the built-in rules searches for Siebel_row_id prefix.
- **Match Case.** Matches the case when looking for boundaries.
- **Use "#" for any digit.** Replaces all digits with a hash sign. The hash signs serve as wildcard, allowing you to find text strings with any digit. For example, if you enable this option and specify **HP###** as the left boundary, **HP193** and **HP284** will be valid matches.

Adding Comments

You can instruct VuGen to insert descriptive comments to the correlation steps within your script. To enable this option, select the **Add Comments to script** option.

Advanced Correlation Rules

VuGen lets you specify the following advanced correlation rules:

- **Always create new parameter.** Creates a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance. This option should be set if the Web server assigns a different value for each page. For example, NetDynamics servers may change the session ID from page to page to minimize fraud.
- **Replace with parameter only for exact matches.** Replace the recorded value with a parameter only when the text between the boundaries exactly matches the found value (from the first snapshot). If there are additional characters either before or after the string, it will not replace the parameter.

For example, in a form submission, VuGen recorded the characters 1234 between the boundaries aaa and bbb, aaa1234bbb. In subsequent submissions of this form, VuGen only replaces the recorded value with a parameter if it finds the characters 1234, Name=1234. If another value is entered, even if it contains the first string, for example, Name=12345, VuGen will not replace the value with a parameter. Instead, it will use the value 12345.

- **Reverse Search.** Searches for the left boundary from the end of the string backwards.
- **Left boundary Instance.** The number of occurrence of the left boundary within the string (not the body) for it to be considered a match.
- **Offset.** The offset of a sub-string of the found value to save to the parameter. The default is the beginning of the matched string. Note that you must specify a non-negative value.
- **Length.** The length from its offset of a sub-string of the matched string to save to the parameter. If you disable this option, the default saves the string from the specified offset until the end of the match.
- **Alternate Right Boundary.** An alternative criteria for the right boundary if the previously specified boundary is not found. You can specify text, **End of String**, or **Newline Character**.

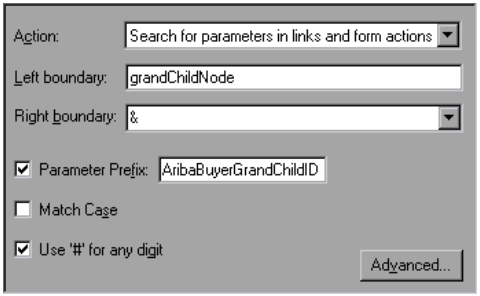
Setting Correlation Rules

You can add, modify, or remove rules using the Correlation Recording options. Note that you can also edit rules that were created automatically for application server environments.

In addition to creating rules using the recording options before recording, you can create rules after recording. After running your script, you scan it for correlations (CTRL+F8). You select one of the correlation results, and create a rule based on its properties. For more information, see "Performing a Scan for Correlations" on page 871.

To define correlation rules:

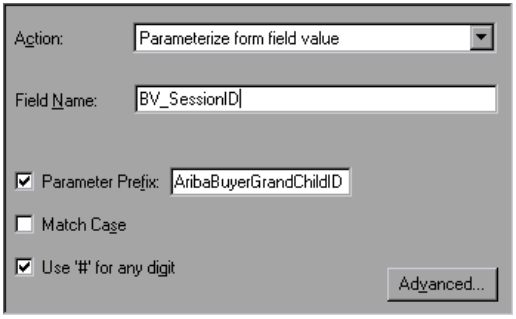
1 Click on an existing rule or click **New Rule** in the left pane. The Correlation Rules are displayed in the right pane.



2 Select a type of action: link or form action, cookie, all body, form field, or web_reg_add_cookie.

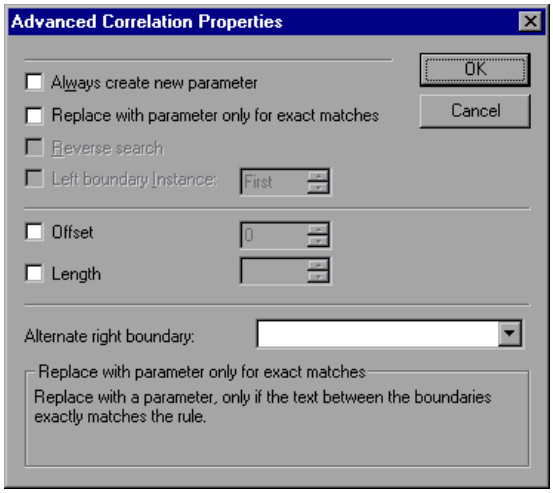
3 For the first three types, specify boundaries of the data in the **Left Boundary** and **Right Boundary** boxes.

4 For form field type actions, specify the field name.



5 Select the desired options: **Match Case** and/or **Parameter Prefix**. Specify a parameter prefix. To convert all digits to hash signs (#), select **Use # for any digit**.

6 To set advanced rules, click **Advanced** in the Correlation node. The Advanced Correlation Properties dialog box opens.



- Select **Always create new parameter** to create a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance.
- Select **Replace with parameter only for exact matches** to replace a value with a parameter only when the text exactly matches the found value.
- Select **Reverse Search** to perform a backward search.
- Select the **Left Boundary Instance** box and specify the desired instance.
- Select **Offset** to specify an offset for the string within the match.
- Select **Length** to specify the length of the matched string to save to the parameter. This option may be used in conjunction with the **Offset** option.
- Specify another right boundary in the **Alternate right boundary** box or select **End of String** or **NewLine Character** from the drop-down menu.

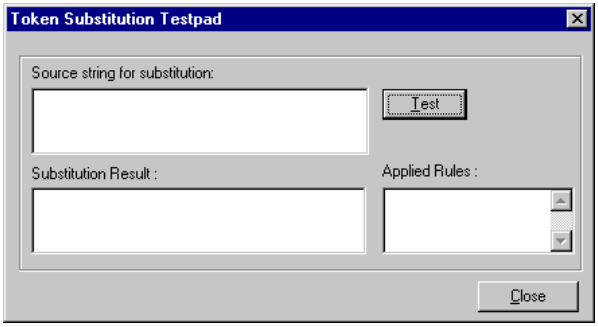
7 Click **Test Rule** to test the rule you just defined. For information, see "Testing Rules" on page 862.

8 Click **OK** to save the rules and close the dialog box.

Testing Rules

This section applies to user-defined rules that you created for a server with a known context. After you define a new rule in the Correlation Rule dialog box, you can test it before recording your session by applying the rules to a sample string. You test the rules in the Token Substitution Testpad. To use the testpad:

1 Select a rule from the left pane and click **Test**. The Token Substitution Testpad dialog box opens.



2 Enter text in the **Source String for Substitution** box.

3 Click **Test**.

If substitution occurred, you will see the parameterized source text in the **Substitution Result** box and a list of rules that were applied to it in the **Applied Rules** box.

54

Web (HTTP/HTML) -Correlation After Recording

When correlation was not performed during recording, VuGen's built-in Web Correlation mechanism allows you to correlate Vuser scripts after a recording session.

This chapter includes:

The following information applies only to Web, Wireless, SAP-Web, and Siebel Web Vuser scripts.

About Correlating with Snapshots

VuGen provides several correlation mechanisms for Web Vuser scripts. The automatic method discussed in Chapter 53, "Web (HTTP/HTML) Correlation Rules" detects dynamic values during recording and allows you to correlate them right away. If you disabled automatic correlation, or if the automatic method did not detect all of the differences, you can use VuGen's built-in correlation mechanism, described in this chapter, to find differences and correlate the values. You can also use this mechanism for scripts that were only partially correlated.

The correlation mechanism uses snapshots to track the results of script execution. Snapshots are graphical representations of Web pages. VuGen captures snapshots of the Web pages during record and replay. You compare the recorded snapshot to any of the replay snapshots to determine which values you need to correlate to successfully run the script. For more information about Record and Replay snapshots, see "Understanding Snapshots" on page 51.

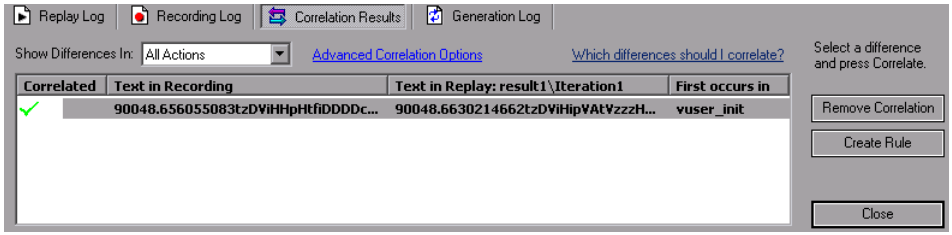
The Web correlation mechanism has a built-in comparison utility that allows you to view the text or binary differences between the snapshots. You can then correlate the differences one-by-one or all at once.

If VuGen's correlation mechanisms are insufficient, or for protocols that do not support these mechanisms, such as Wireless, use manual correlation. For more information, see "Performing Manual Correlation" on page 875.

Viewing the Correlation Results Tab

The Correlation Results tab displays the differences between the Record and Replay snapshots.

When you instruct VuGen to scan the script for correlations, it opens the Output window and displays the differences between the recording and replayed snapshots in the **Correlation Results** tab.



You can display all the differences in the script or only those for the current step or action, by selecting the desired option from the **Show Differences In** list box.

Differences that were correlated are indicated by a check mark in the **Correlated** column. The next two columns, **Text in Recording**, **Text in Replay** show the text differences between the snapshots. The next column, **First occurs in**, indicates the Action in which the correlation was first detected.

After you detect the differences between the snapshots, you correlate them one at a time by selecting the correlation and clicking **Correlate**. VuGen also allows you to undo a specific correlation using the **Remove Correlation** button. If you expect one of the detected correlations to occur in subsequent recordings, you can create a new correlation rule. By creating rules, you enable VuGen to recognize differences during recording and automatically correlate them. For more information, see "Creating a Rule" on page 866.

When you correlate a value using the this mechanism, VuGen inserts a web_reg_save_param function and a comment into your script indicating that a correlation was done for the parameter. It also indicates the original value.

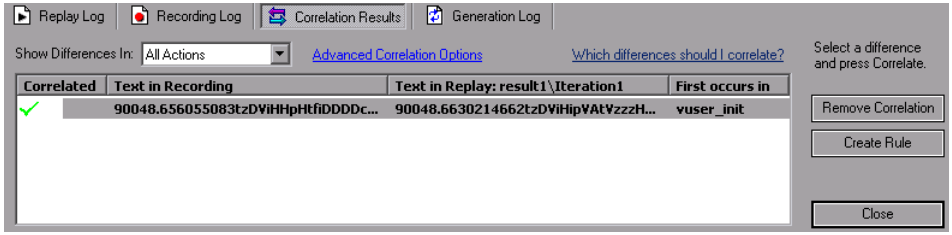
```
// [WCSPARAM WCSParam_Diff1 14 reserveFlights] Parameter {WCSParam_Diff1}
created by Correlation Studio
    web_reg_save_param("WCSParam_Diff1", "LB= NAME=\"", "RB=\"", "Ord=5",
"Search=Body", "RelFrameId=1", LAST);
    web_submit_form("reservations.pl",
        "Snapshot=t4.inf",
        ITEMDATA,
        "Name=depart", "Value=Denver", ENDITEM,
        "Name=departDate", "Value=06/25/2004", ENDITEM,
        "Name=arrive", "Value=Los Angeles", ENDITEM,
        "Name=returnDate", "Value=06/26/2004", ENDITEM,
        "Name=numPassengers", "Value=1", ENDITEM,
        "Name=roundtrip", "Value=<OFF>", ENDITEM,
        "Name=seatPref", "Value=None", ENDITEM,
        "Name=seatType", "Value=Coach", ENDITEM,
        "Name=findFlights.x", "Value=44", ENDITEM,
        "Name=findFlights.y", "Value=12", ENDITEM,
        LAST);
    lr_think_time(12);
```

Creating a Rule

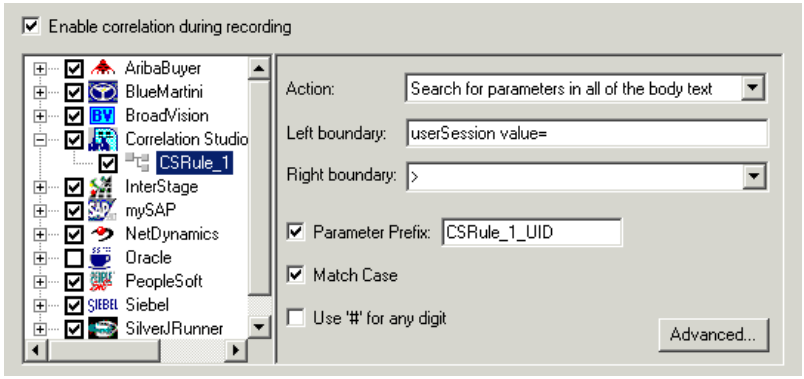
You can create a rule directly from the list of Correlated Results. Creating a rule, enables VuGen to recognize the difference during recording and automatically correlate it.

To create a rule from one of the detected correlations:

Select the correlation and click **Create Rule**. You can also create a rule by selecting a correlation and choosing **Create Correlation Rule** from the right-click menu.

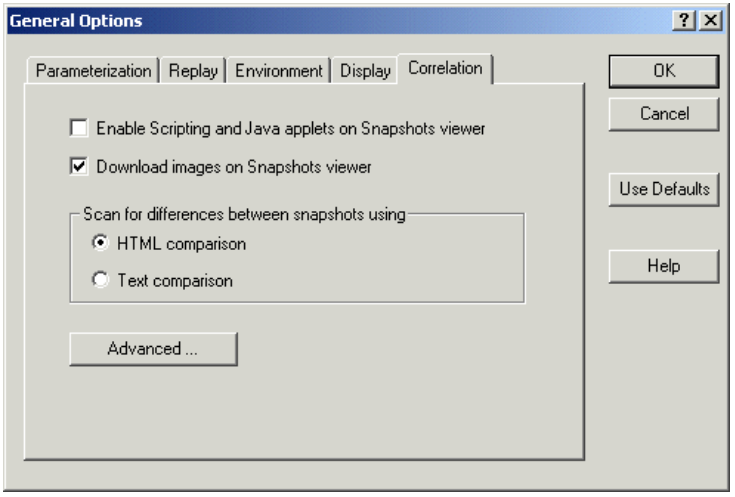


VuGen adds this rule to the list of **Correlation** rules. You can view this rule in the Recording Options Correlation node. In the following example, VuGen added the rule as CSRule_1.



Setting Up VuGen for Correlations

You set the global Correlation setting under the General options. These options instruct the Vusers to save correlation information during replay, to be used at a later stage. You can specify the type of comparison to perform when comparing snapshots: HTML or text. In the Advanced options, you can indicate which characters should be treated as delimiters.



- **Enable Scripting and Java applets on Snapshots viewer.** Allows VuGen to run applets and JavaScript in the snapshot window. This is disabled by default because it uses a lot of resources.
- **Download images on Snapshots viewer.** Instructs VuGen to display graphics in the Snapshot view. If you find that the displaying of images in the viewer is very slow, you can disable this option. This option is enabled by default.
- **Scan for differences between snapshots using.** Select a comparison method:
  - **HTML Comparison.** Only display the differences in HTML code.
  - **Text Comparison.** Display all text, HTML, and binary differences.

Note: In most cases, we recommend that you work with the default HTML comparison method. If your script contains non-HTML tags, you can use the Text comparison method.

➤ **Advanced.** Opens the Advanced Correlation dialog box.

Advanced Correlation dialog box

This dialog box lets you specify the characters to be treated as delimiters.

➤ **Characters that should be treated as delimiters.** Specifies one or more non-standard delimiters.

➤ **Additional Delimiters.** You can specify standard delimiters such as Carriage Return, New line and Tab characters. To change this setting, clear the check box next to the delimiter.

➤ **Ignore differences shorter than … characters.** Allows you to specify a threshold for performing correlation. When VuGen compares the recorded data with the replay data during the scanning process, it detects differences. It will not correlate the differences unless the number of different characters is greater than or equal to the threshold value. The default value is 4 characters.

➤ **Issue a warning for large correlations.** Issues a warning if you try to correlate a string whose size is 10 KB or larger.

Setting the Correlation Preferences

Before recording a session, you configure the correlation preferences.

To set the correlation preferences:

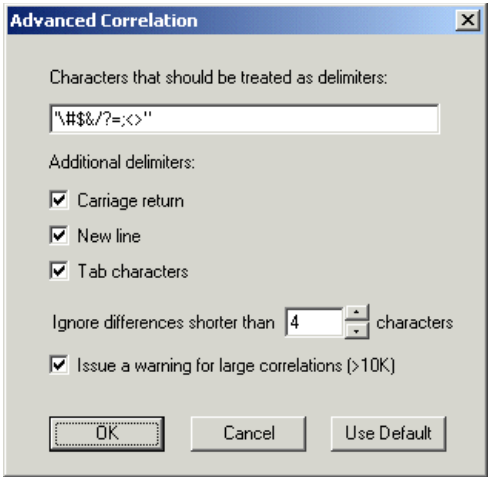
1 Select **Options** > **General** and select the **Correlation** tab.

2 Select **Enable Scripting and Java applets on Snapshots viewer** to allow VuGen to run applets and JavaScript in the snapshot window.

3 To instruct VuGen to display graphics in the Snapshot view, select the **Download images on Snapshots viewer** option.

4 Select the comparison method: **HTML comparison** or **Text Comparison** (for non-HTML elements only).

5 To set the delimiter characters, click **Advanced** to open the Advanced Correlation dialog box.



6 In the **Characters that should be treated as delimiters** box, specify all characters that are to be treated as delimiters.

7 Select the desired options in the **Additional delimiters** section, to specify one or more standard delimiters.

8 Specify a threshold for the correlation in the **Ignore differences shorter than** box. When VuGen compares the recorded data with the replay data during the scanning process, it detects differences. It will not correlate the differences unless the number of different characters is greater than or equal to the threshold value.

9 To **issue a warning for large correlations**, select the option's check box.

10 Click **OK** to accept the Advanced Correlation settings and close the dialog box.

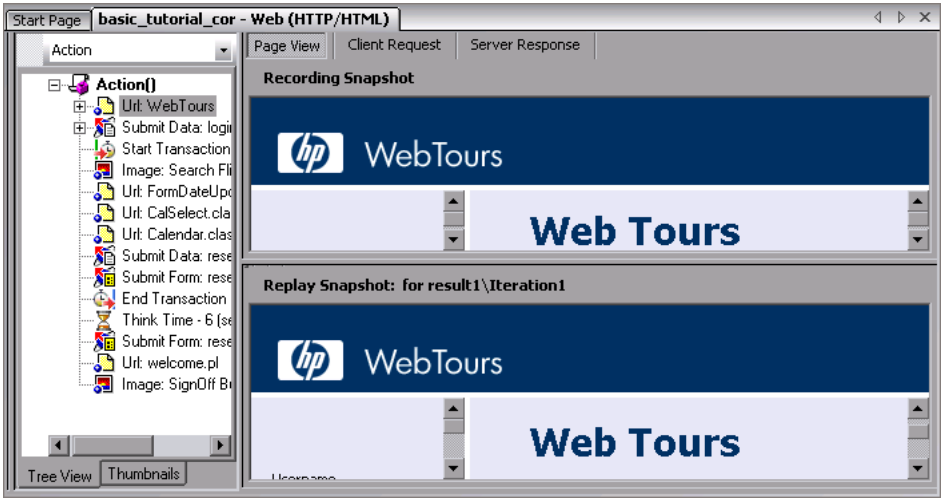
11 Click **OK** in the General Options dialog box to accept the Correlation setting and close the dialog box.

Performing a Scan for Correlations

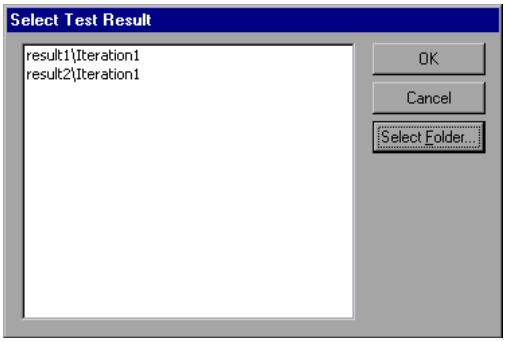
You can use VuGen's snapshot window to determine which values within your script are dynamic and require correlation. The following section describes how to automatically scan the script for differences and use VuGen to perform the necessary correlations.

To scan your script for correlations:

1. Open a script and view it in Tree view (**View > Tree View).** Display the snapshots (**View > Snapshot > View Snapshot).**
2. Select a script step in the Tree view from the left pane. A snapshot opens in the right pane.
3. To display both the recording snapshot and the first replay snapshot, click **View > Snapshot > Recorded and Replayed**.

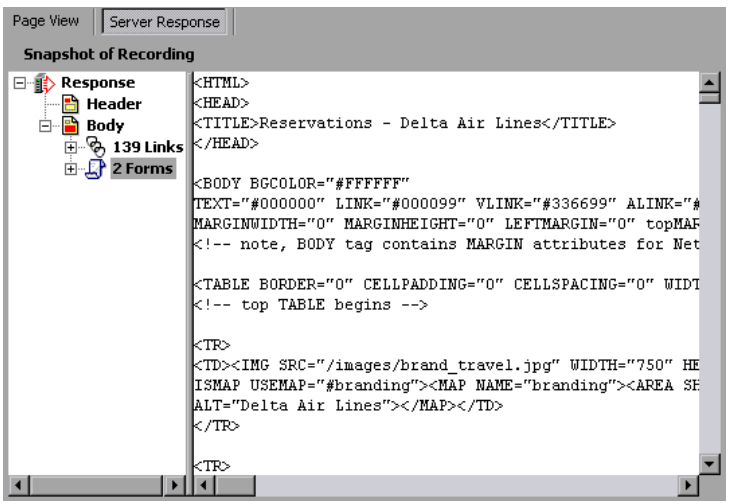


4 To use a snapshot other than the first, click **View** > **Snapshot** > **Select Iteration.** A dialog box opens, displaying the folders that contain snapshot files. These are usually the **result** and **Iteration** folders below the script's folder.



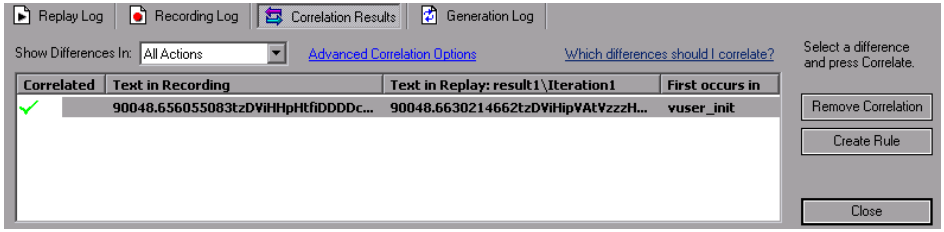
5 To select a snapshot file in a folder other than the one in the subfolders of the script, click **Select Folder**. Browse to the desired location, and click **OK**.

6 To view the HTML code, click the **Server Response** tab. Expand the Body branch.



To return to the page view, click the **Page View** tab.

7 Select **Vuser** > **Scan for Correlations** or click the **Find Correlations** button. VuGen scans the script for dynamic values that need to be correlated and displays them in the Correlation Results tab.



8 View all differences or select a filter method in the **Show Differences In** list box. The options are **All Actions**, **Current Action**, or **Current Step Only**.

Determining the Differences to Correlate

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

The following strings most probably require correlation:

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

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

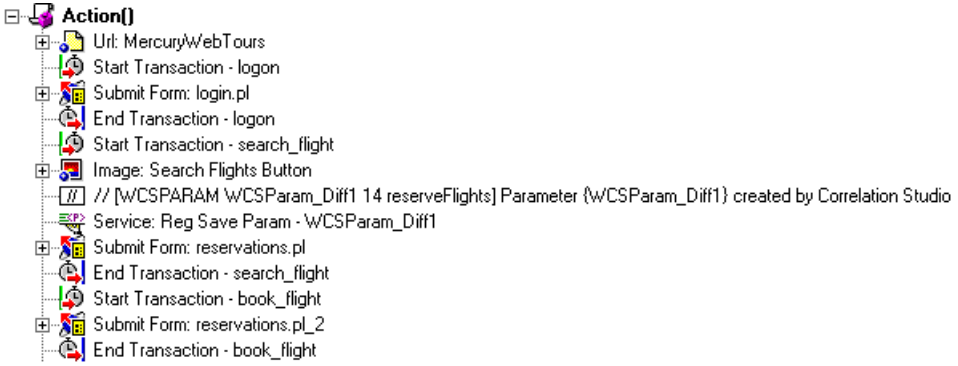
You should also correlate differences in which some of the recorded and replayed strings are identical, but others differ. For example, SessionID strings with identical prefixes and suffixes, but different characters in between, should be correlated.

Once you determine that a difference needs to be correlated, you instruct VuGen to correlate it.

To correlate the differences:

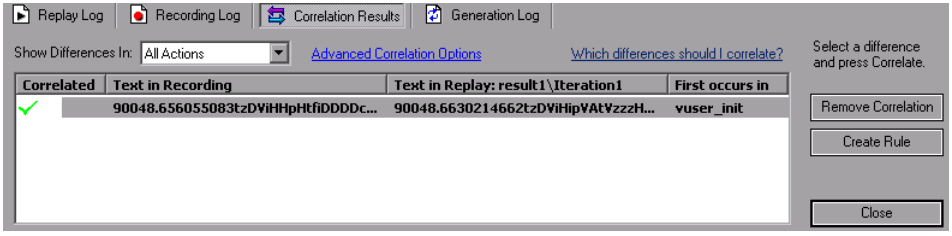
1 View the differences in the Correlation tab, and select the one you want to correlate. We recommend that you correlate only one difference at a time.

2 Click **Correlate**. VuGen places a green check mark next to differences that were correlated and inserts a **web_reg_save_param** function into the script.

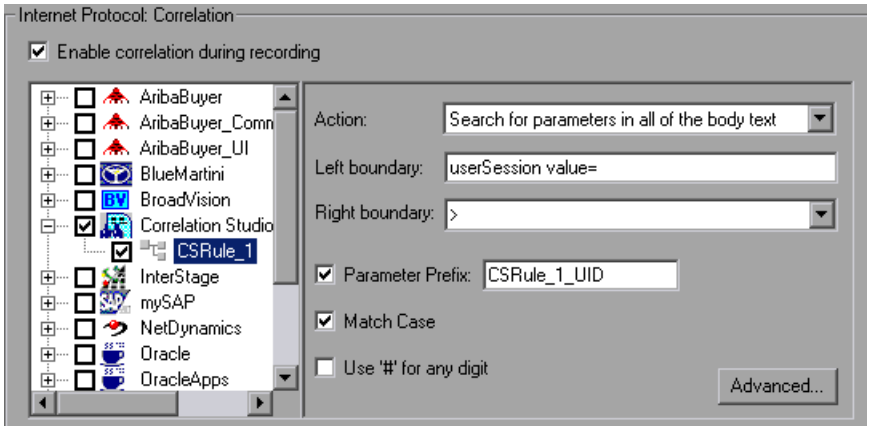


Repeat this step for all differences you want to correlate.

3 To create a rule from one of the detected correlations, select the correlation and click **Create Rule**. This is also available from the right-click menu. VuGen issues a message confirming that your rule was created.



To view this rule, open the Recording Options (CTRL +F7) and select the **Correlation** node. Expand the **Correlation Studio** entry and select your rule.



4 To undo a correlation, select the difference and click **Remove Correlation.**

5 Select **File** > **Save** to save the changes to your script.

Performing Manual Correlation

For Web Vusers, VuGen's automatic or rule-based correlation usually correlates the scripts dynamic functions so that you can run the script successfully. You can also perform correlation after the recording session using VuGen's snapshot comparison.

For Wireless Vusers and other Vuser scripts for which automatic correlation did not apply, VuGen also allows you to manually correlate your scripts. You manually correlate a script by adding the code correlation functions. The function that allows you to dynamically save data to a parameter is **web_reg_save_param.**

When you run the script, the **web_reg_save_param** function scans the subsequent HTML page that is accessed. You specify a left and/or right boundary and VuGen searches for text between those boundaries. When VuGen finds the text, it assigns it to a parameter.

The function's syntax is as follows:

```
int web_reg_save_param (const char *mpszParamName, <List of Attributes>, LAST);
```

The following table lists the available attributes. Note that the attribute value strings (for example, Search=all) are not case sensitive.

NotFound	The handling method when a boundary is not found and an empty string is generated. "ERROR," the default, indicates that VuGen should issue an error when a boundary is not found. When set to "EMPTY," no error message is issued and script execution continues. Note that if Continue on Error is enabled for the script, then even when NOTFOUND is set to "ERROR," the script continues when the boundary is not found, but it writes an error message to the Extended log file.
LB	The left boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.
RB	The right boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.
RelFrameID	The hierarchy level of the HTML page relative to the requested URL. The possible values are ALL or a number.
Search	The scope of the search—where to search for the delimited data. The possible values are Headers (search only the headers), Body (search only Body data, not headers), or ALL (search Body and headers). The default value is ALL.
ORD	This optional parameter indicates the ordinal or occurrence number of the match. The default ordinal is 1. If you specify "All," it saves the parameter values in an array.

SaveOffset	The offset of a sub-string of the found value, to save to the parameter. The default is 0. The offset value must be non-negative.
Savelen	The length of a sub-string of the found value, from the specified offset, to save to the parameter. The default is -1, indicating until the end of the string.
Convert	The conversion method to apply to the data: HTML_TO_URL: convert HTML-encoded data to a URL-encoded data format HTML_TO_TEXT: convert HTML-encoded data to plain text format

To manually correlate your script:

1. Identify the statement that contains dynamic data and the patterns that characterize the boundaries of the data. See "Defining a Dynamic String's Boundaries" on page 880.
2. In the script, replace the dynamic data with your own parameter name. See below for more details.
3. Add the **web_reg_save_param** function into the script before the statement that contains the dynamic data. See "Adding a Correlation Function" on page 878 or the *Online Function Reference* (**Help > Function Reference**).

Replacing Dynamic Data with a Parameter

Identify the actual dynamic data in the recorded statement, then search the entire script for the dynamic data and replace it with a parameter. Give the parameter any name and enclose it with braces: {param_name}. You can include a maximum of 64 parameters per script.

To replace dynamic data with a parameter:

Select **Edit** > **Replace** from the VuGen main window to display the Search and Replace dialog box. Search the entire script for the dynamic data and replace it with a parameter.

Adding a Correlation Function

You insert the **web_reg_save_param** statement to save dynamic data in a script. This function tells VuGen to create a parameter that saves the run-time value of the dynamic data during replay.

When you run the script, the **web_reg_save_param** function scans the subsequent HTML page that is accessed. It searches for an occurrence of the left boundary, followed by any string, followed by the right boundary. When such an occurrence is found, VuGen assigns the string between the left and right boundaries to the parameter named in the function's argument. After finding the specified number of occurrences, **web_reg_save_param** does not search any more HTML pages. The Vuser continues with the next step in the script.

Sample Correlation for Web Vusers

Suppose the script contains a dynamic session ID:

```
web_url("FirstTimeVisitors",
    "URL=/exec/obidos/subst/help/first-time-visitors.html/002-8481703-
        4784428>Buy books for a penny ",
    "TargetFrame=",
    "RecContentType=text/html",
    "SupportFrames=0",
     LAST);
```

You insert a **web_reg_save_param** statement before the above statement:

```
web_req_save_param ("user_access_number", "NOTFOUND=ERROR", "LB=first-
time-visitors.html/","RB=>Buy books for a penny", "ORD=6", LAST);
```

After implementing correlated statements, the modified script looks like this, where user_access_number is the name of the parameter representing the dynamic data.

```
web_url("FirstTImeVisitors",
    "URL=/exec/obidos/subst/help/first-time-”
       "visitors.html/{user_access_number}Buy books for a penny",
    "TargetFrame=",
    "RecContentType=text/html",
    "SupportFrames=0",
     LAST);
```

Note: Each correlation function retrieves dynamic data once, for the subsequent HTTP request. If another HTTP request at a later point in the script generates new dynamic data, you must insert another correlation function.

Sample Correlation for Wireless Vusers

Suppose your script contains a dynamic session ID for a WAP connection:

```
web_url("login.po;sk=IuZSuuRlHUMnpF-wpK8PzEpy(1YOSBSMy)",
    "URL=http://room33.com/portal/login.po;sk=IuZSuuRlHUMnpF-
        wpK8PzEpy(1YOSBSMy)",
    "Resource=0",
    "RecContentType=text/vnd.wap.wml",
    "Mode=HTML",
    LAST);
```

You insert a web_reg_save_param statement before the above statement and replace the dynamic value with the parameter. In the following example, the web_reg_save_param functions saves the login ID string to a variable called SK. It saves binary data, denoted by the RB/BIN attribute, and sets the left boundary as "sk=".

```
web_reg_save_param(
    "SK",
"LB=sk=",
"RB/BIN=#login\\x00\\x01\\x03",
"Ord=1",
 LAST);


web_url("login.po;sk={SK}",
     "URL=http://room33.com/portal/login.po;sk={SK}",
     "Resource=0",
     "RecContentType=text/vnd.wap.wml",
     "Mode=HTML",
     LAST);
```

Defining a Dynamic String's Boundaries

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

- Always analyze the location of the dynamic data within the HTML code itself, and not in the recorded script.
- Identify the string that is immediately to the left of the dynamic data. This string defines the left boundary of the dynamic data.
- Identify the string that is immediately to the right of the dynamic data. This string defines the right boundary of the dynamic data.

- **web_reg_save_param** looks for the characters between (but not including) the specified boundaries and saves the information beginning one byte after the left boundary and ending one byte before the right boundary. **web_reg_save_param** does not support embedded boundary characters.
  For example, if the input buffer is {a{b{c} and "{" is specified as a left boundary, and "}" as a right boundary, the first instance is c and there are no further instances—it found the right and left boundaries but it does not allow embedded boundaries, so "c" is the only valid match.

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

```
web_set_max_html_param_len("1024");
```

55

Web (HTTP/HTML) - Handling XML Pages

VuGen's Web Vusers support Web pages containing XML code.

This chapter includes:

About Testing XML Pages

VuGen supports record and replay for XML code within Web pages.

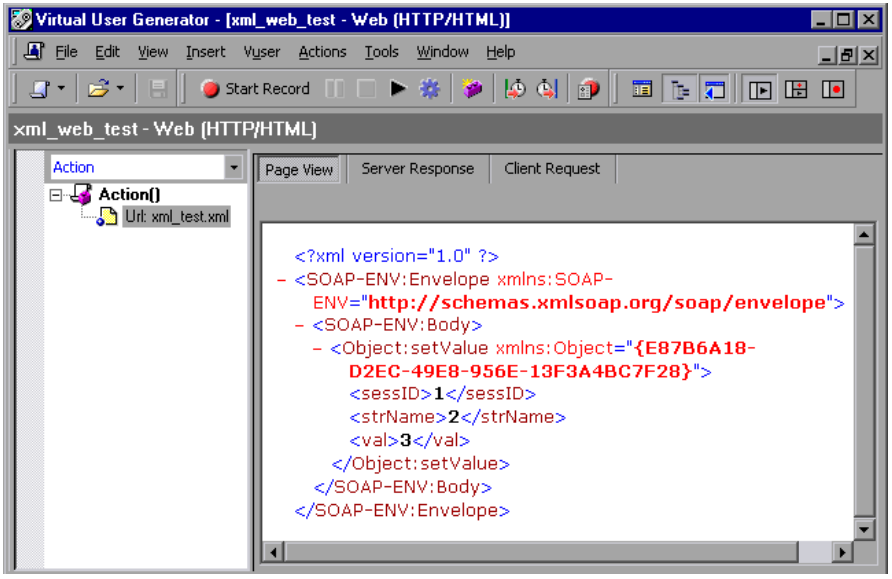
The XML code can appear in the script as a regular URL step or as a custom request. VuGen detects the HTML and allows you to view each document type definition (DTD), its entities, and its attributes. VuGen can interpret the XML when the MIME type displayed in the **RecContentType** attribute or the MIME type returned by the server during replay, ends with **xml**, such as **application/xml** or **text/xml**. The DTD is color coded, allowing you to identify each one of the elements. You can also expand and collapse the tree view of the DTD.

When you expand the DTD, you can parameterize the attribute values. You can also save the values in order to perform correlation using the standard correlation functions. For more information about the correlation functions, see the *Online Function Reference* (**Help** > **Function Reference**).

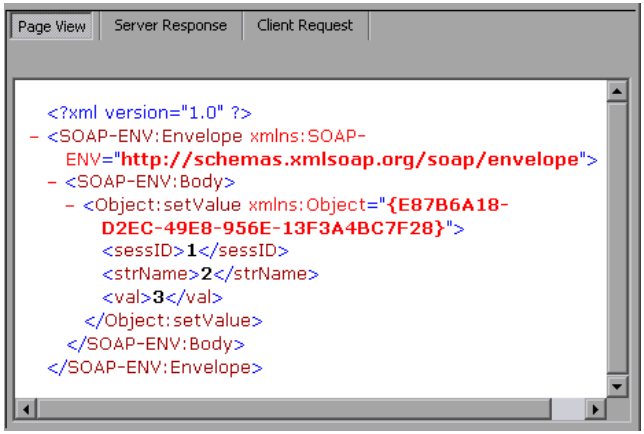
Note: VuGen cannot display a DTD with **XML islands**, segments of XML embedded inside an HTML page. VuGen only displays pages that are entirely XML.

Viewing XML as URL Steps

One way to test a page with XML code, is to record it with VuGen. You record the XML pages as you would record a standard Web page. VuGen records the DTD and all of the XML elements. It does not create a snapshot for the XML page. Instead, for each XML step it displays the XML code in the snapshot frame under the Server Response tab.

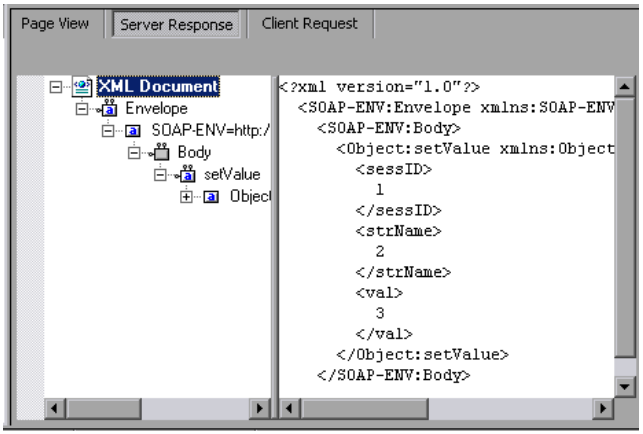


VuGen creates a color-coded expandable hierarchy of the DTD in the snapshot frame. Click on the "+" to expand an item, and click on the "-" to collapse it. VuGen displays all XML tags in brown, and values in black.

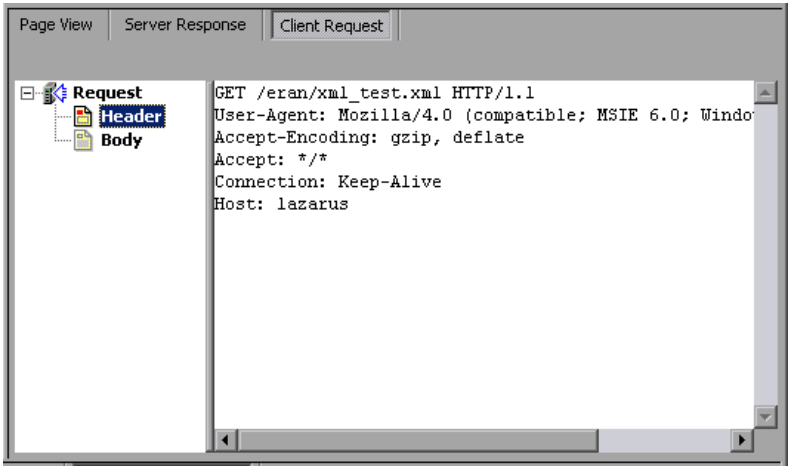


To replace any of the constant values with a parameter, select a value, perform a right-click, and select **Replace with a Parameter**. Follow the standard procedure for parameterization. For more information, see Chapter 70, "Working with VuGen Parameters."

You can also view the Server response and Client request for the XML page by clicking the appropriate tab. The following example shows the Server response of an XML page. Note that you can expand and collapse all branches of the XML tree.



The following example shows the Client Request for the header of an XML page:

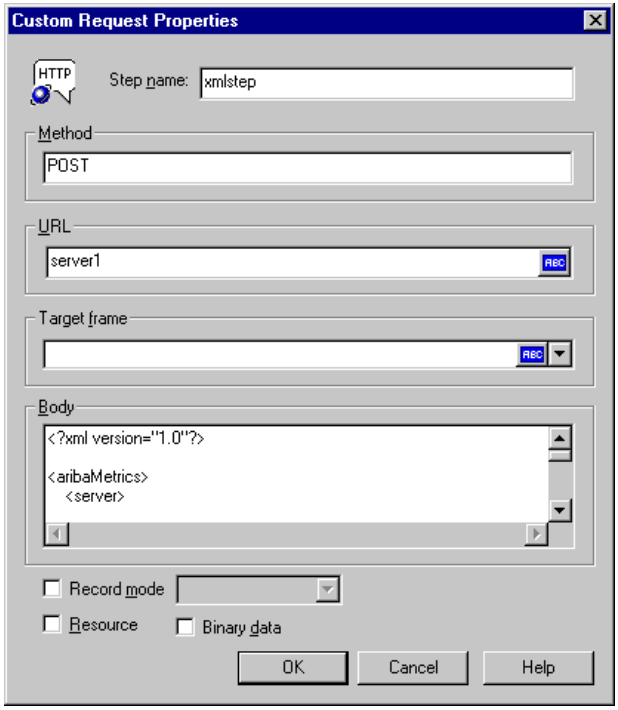


Inserting XML as a Custom Request

You can also test your XML pages by inserting the XML code as a custom request. In this mode, the **Custom Request** properties box displays the elements of the DTD in either text or XML format.

To add XML code as a Custom Request:

1. View the script in tree view mode, place the cursor at the desired location, and select **Insert** > **Add Step**. The Add Step dialog box opens.
2. Scroll to the bottom of the list and select **Custom Request**. Click **OK**. The Custom Request Properties dialog box opens.
3. Enter a step name, method (GET or POST), URL, and target frame (optional).
4. Copy the XML code from your browser or editor and paste it into the **Body** section of the Custom Request Properties box.



5 Select the applicable replay options: **Record mode**, **Resource**, or **Binary data**. For more information, see Chapter 52, "Modifying Web and Wireless Vuser Scripts."

6 Click **OK**. VuGen places the custom request step into your script.

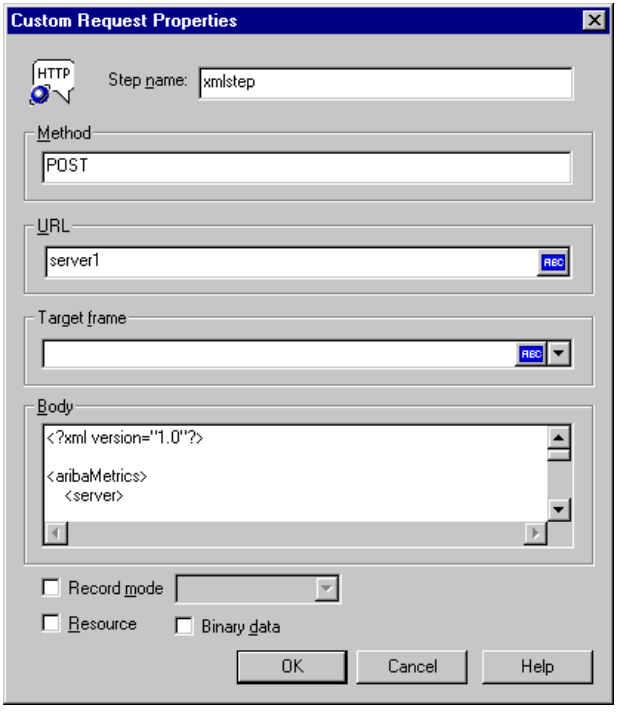
Viewing XML Custom Request Steps

You can view or modify the XML code implemented as a custom request step, at any time. VuGen provides a viewer that allows you to view the hierarchy of the DTD, and expand and collapse the elements as needed.

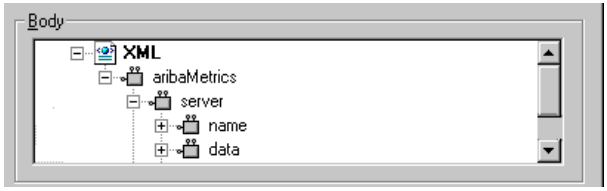
To view the XML code of a custom request step:

1 View the script in tree view mode, and select the desired step.

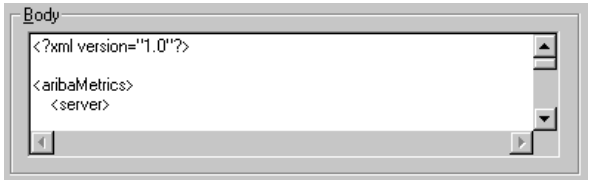
2 Select **Properties** from the right-click menu. The Custom Request Properties dialog box opens.



The bottom section of the dialog box displays the XML code. If the **RecContentType** attribute is set to **text/xml**, by default VuGen displays the code in an XML format hierarchy. In this mode, the XML code is not editable.

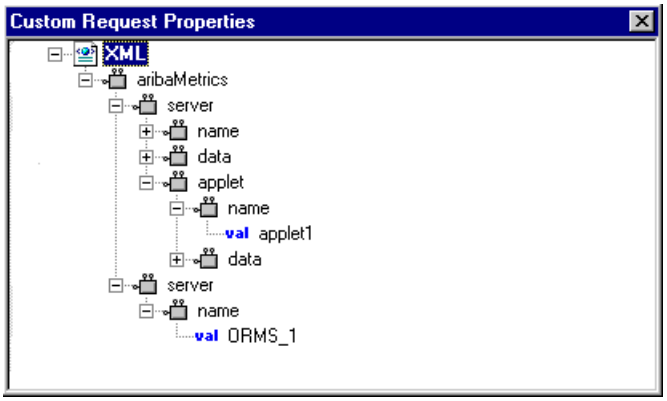


If the **RecContentType** attribute is set to any type other than **text/xml**, VuGen displays the code in plain text format. In this mode, the XML code is editable.



3 To switch between the text and XML views, select **XML view** or **Text view** from the right-click menu.

4 When you are in XML view, you can view the code in a larger window. Select **Extended view** from the right-click menu. To switch back to the dialog box view, select **Normal view** from the right-click menu.



56

Oracle NCA Protocol

You can use VuGen to create scripts that emulate an Oracle NCA user. You record typical NCA business processes with VuGen. You then run the script to emulate users interacting with your system.

This chapter includes:

About Creating Oracle NCA Vuser Scripts

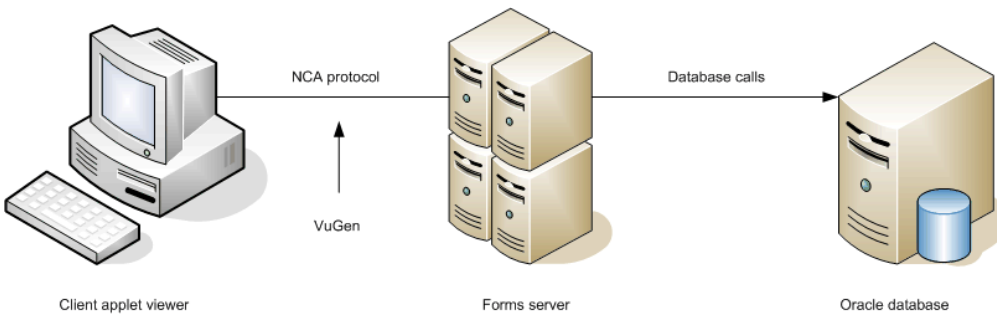
Oracle NCA is a Java-based database protocol. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer.

This eliminates the need for client software and allows you to perform database actions from all platforms that support the applet viewer. There is a Vuser type specifically designed to emulate an Oracle NCA client.

The NCA environment is a three-tier environment. The user first sends an http call from his browser to a Web server. This call accesses the startup HTML page which invokes the Oracle Applications applet. The applet runs locally on the client machine—all subsequent calls are communicated between the client and the Forms server through the proprietary NCA protocol.

The client (applet viewer) communicates with the application server (Oracle Forms server) which then submits information to the database server.

VuGen records and replays the NCA communication between the client and the Forms server (application server).



When you record an Oracle NCA session, VuGen records all of the NCA and Web actions, even if you only created a single protocol script. If you know in advance that the Web functions are important for your test, create a multi-protocol script from the beginning for the Oracle NCA and Web protocols.

If you initially created a single protocol script for Oracle NCA, and at a later stage you require the Web functions for testing, you can regenerate your script in VuGen to add the Web functions, without having to re-record the session. You indicate this from the Protocols node in the Regenerate Script dialog box. For more information, see Chapter 5, "Recording with VuGen."

Getting Started with Oracle NCA Vusers

The following procedure outlines how to create an Oracle NCA Vuser script.

1 Make sure that the recording machine is properly configured.

Make sure that your machine is configured to run the Oracle NCA applet viewer, before you start VuGen. You must also make sure VuGen supports your version of Oracle Forms. For more information, see "Recording Guidelines" on page 894 and the Readme file.

2 Create a skeleton Oracle NCA Vuser script.

Use VuGen to create a skeleton Oracle NCA Vuser script. For details, see Chapter 5, "Recording with VuGen."

3 Record typical user actions.

Begin recording, and perform typical actions and business processes from the applet viewer. VuGen records your actions and generates a Vuser script. For details, see Chapter 5, "Recording with VuGen."

4 Enhance the Vuser script.

Use the Insert menu to add transactions, rendezvous points, comments, and messages in order to enhance the Vuser script. For details, see Chapter 6, "Enhancing Vuser Scripts."

5 Parameterize the script.

Replace recorded constants with parameters. For details, see Chapter 70, "Working with VuGen Parameters."

6 Set the run-time properties for the script.

Configure run-time settings for the Vuser script. The run-time settings define certain aspects of the script execution. For details, see Chapter 79, "Configuring Run-Time Settings."

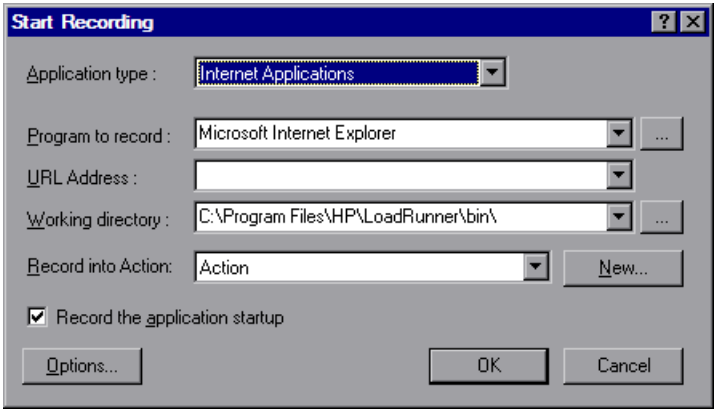
7 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

Recording Guidelines

When recording an Oracle NCA Vuser script, follow these guidelines:

➤ Specify which browser VuGen should use when recording an Oracle NCA session. In the Start Recording dialog box, select the desired browser in the **Program to Record** list. The list contains all of the available browsers.



➤ Close all browsers before you begin recording.

➤ Record the login procedure in the vuser_init section. Record a typical business process in the Actions section. When you run the script, you can then specify multiple iterations for a specific business process. For more information, see "Creating New Virtual User Scripts" on page 94.

```
vuser_init()
{
    nca_set_connect_opt(SCALE_INFO, 11, 18);
    nca_connect_server("labm1orcl05.devlab.ad", "9000",
"module=/opt/applvis/visappl/fnd/11.5.0/forms/US/FNDSCSGN
userid=APPLSYSPUB/PUB@VIS fndnam=APPS record=names  ");
    nca_set_window("Oracle Applications");
    nca_edit_set("SIGNON_USERNAME_0", "OPERATIONS");
    nca_obj_type("SIGNON_USERNAME_0", '\t', 0);
    nca_edit_set("SIGNON_PASSWORD_0", lr_decrypt("4768d647f4f1840f2e46d5"));
    nca_button_press("SIGNON_CONNECT_BUTTON_0");
return 0;
}
```

➤ Due to a Netscape limitation, you cannot launch an Oracle NCA session within Netscape when another Netscape browser is already running on the machine.

➤ VuGen supports the recording of Oracle Forms applications using the Forms Listener Servlet in multi--protocol mode. The application server uses the **Forms Listener Servlet** to create a runtime process for each client. The runtime process, **Forms Server Runtime**, maintains a persistent connection with the client and sends information to and from the server.

To support Forms 4.5 in replay, set the following in the **mdrv.dat** file:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp110.dll
WIN95_EXT_LIBS=ncarp110.dll
LINUX_EXT_LIBS=liboranca.so
SOLARIS_EXT_LIBS=liboranca.so
HPUX_EXT_LIBS=liboranca.sl
AIX_EXT_LIBS=liboranca.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api
```

To restore Forms support for versions later than 4.5, restore the original values.

Enabling the Recording of Objects by Name

When recording an Oracle NCA script, you must record the session using object names instead of the standard object ID. If the script is recorded using the object ID, replay will fail because the ID is generated dynamically by the server and differs between record and replay. You can verify that your script is being recorded with object names by examining the **nca_connect_server** statement.

```
nca_connect_server("199.35.107.119","9002"/*version=11i*/,"module=/d1/oracle/visap
pl/fnd/11.5.0/forms/US/FNDSCSGN userid=APPLSYSPUB/PUB@VIS fndnam=apps
record=names ");
```

If the **record=names** argument does not appear in the **nca_connect_server** function, you are recording object IDs. You can instruct VuGen to record object names in by modifying one of the following:

- Startup HTML File
- URL to Record
- Forms Configuration File

Note that the ability to capture the developer name for all objects was introduced in Oracle Forms6i Patch 9 (Oracle Forms Version: 6.0.8.18.3). Test Starter Kit scripts that were written before the release of Oracle Forms 6i Patch 9 will not have the developer name as part of an object's physical description, except for the edit fields.

Startup HTML File

If you have access to the startup HTML file, you instruct VuGen to record object names instead of its object ID by setting the **record=names** flag in the startup file, the file that is loaded when you start the Oracle NCA application.

Edit the startup file that is called when the applet viewer begins. Modify the line:

```
<PARAM name="serverArgs … fndnam=APPS">
```

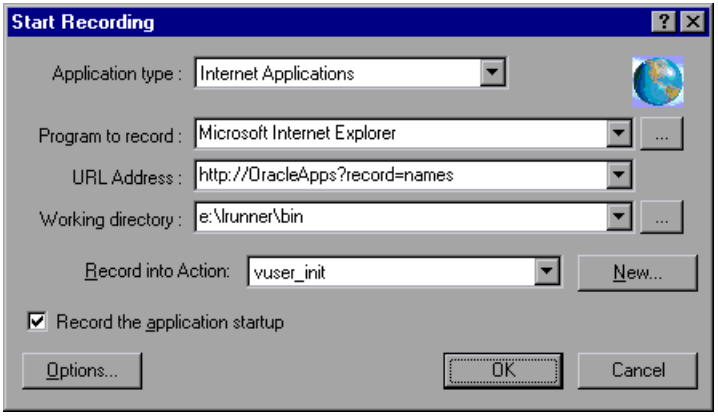
and add the Oracle key "record=names":

```
<PARAM name="serverArgs … fndnam=APPS record=names">
```

URL to Record

If you do not have access to the startup HTML file, you can still have Oracle NCA record object names instead of its object ID by modifying the URL to record. The following solution only works if the startup HTML file does not reference another file while loading.

For this solution, you add "**?record=names**" after the URL in the Start Recording dialog box, after the URL name to record. This allows VuGen to record object names for the session.



Forms Configuration File

If the application has a startup HTML file that references a Forms Web CGI configuration file **formsweb.cfg** (a common reference), you may encounter problems if you add **record=names** to the Startup file.

In this situation, you should modify the configuration file.

To modify the configuration file to record object names:

1 Go to the Forms Web CGI configuration file.

2 Define a new parameter in this file (see sample Web CGI configuration file below for this change).

```
serverApp=forecast
serverPort=9001
serverHost=easgdev1.dats.ml.com
connectMode=socket
archive=f60web.jar
archive_ie=f60all.cab
xrecord=names
```

3 Open the startup HTML file and locate PARAM NAME="serverArgs".

4 Add the variable name as an argument to the ServerArgs parameter, for example, record=%xrecord%.

```
<PARAM NAME="serverArgs" VALUE="module=%form% userid=%userid%
%otherParams% record=%xrecord%">
```

5 Alternatively, you can edit the **basejini.htm** file in Oracle Forms installation directory. This file is the default HTML file for running a form on the web using JInitiator-style tags to include the Forms applet. In the basejinin.hmt file add the following line to the parameter definitions:

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the <EMBED> tag, add the following line:

```
serverApp="%serverApp%"
logo="%logo%"
imageBase="%imageBase%"
formsMessageListener="%formsMessageListener%"
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file **formsweb.cfg**, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the **basejini.htm** file and store it at another location. In the servlet configuration file, edit the **baseHTMLJinitiator** parameter to point to the new file.

Oracle Applications via the Personal Home Page

When launching Oracle Forms 6i applications by logging in through the **Personal Home Page,** you must set several system profile options at the user level. It is desirable to pass such variables at the user level, and not at the site level, where it will affect all users.

To configure the "ICX: Forms Launcher" profile:

1 Sign on to the application and select the "System Administrator" responsibility.

2 Select **Profile/System** from the Navigator menu.

3 Within the **Find System Profile Values** form:

   a Select the **Display:Site** option

   b **Users** = <your user logon> (i.e. operations, mfg, and so on)

   c **Enter Profile** =%ICX%Launch%

   d Click **Find**.

4 Update the User value to the **ICX:Forms Launcher** profile:

   ➤ If no parameter has been passed to the URL, append the following string to the end of the URL of the user value: **?play=&record=names**

   ➤ If a parameter has been passed to the URL, append the following string to the end of the URL of the user value: **&play=&record=names**

5 Save the transaction.

6 Log out of the Oracle Forms session.

7 Log out of the Personal Home Page session.

8 Sign on again via the **Personal Home Page** using your username.

If you were unable to update the ICX: Forms Launcher profile option at the user level, open the **Application Developer** responsibility and select the **Updatable** option for the ICX_FORMS_LAUNCHER profile.

The first parameter passed to the URL, must begin with a question mark (?). You pass all subsequent parameters with an ampersand (&). In most cases, the URL already contains parameters, which you can identify by searching for a question mark.

Using Oracle NCA Vuser Functions

VuGen records typical NCA business processes and generates Oracle NCA-specific functions. The functions use an **nca** prefix.

The NCA functions are divided into the following categories: Button Object, Connection, Combo Box Object, Edit and Edit Box Object, Flexfield Window, Java Object, List Object, Menu Object, Message Object, Object, Response Object, Scroll Object, Session, Tab Object, Tree Object, and Window Object Functions.

You can also manually program any of the functions into your Vuser script. In text view, you can manually add new functions utilizing the Intellisense and Complete Function features. In Tree view, select **Insert** > **New Step** and select the desired step.

For more information about the Oracle NCA Vuser functions, see the *Online Function Reference* (**Help** > **Function Reference**).

You can further enhance your script with C Vuser functions such as **lr_output_message** and **lr_rendezvous**. For information on using these functions, see Chapter 6, "Enhancing Vuser Scripts."

Understanding Oracle NCA Vusers

When you create an Oracle NCA Vuser script, VuGen records all of the NCA communication between the client and the application server. While you record, VuGen generates context sensitive functions. These functions describe your actions on the database in terms of GUI objects (such as windows, lists, and buttons). As you record, VuGen inserts the context sensitive functions into the Vuser script.

After you finish recording, you can modify the functions in your script, or add additional functions to enhance it. For information about enhancing Vuser script, see Chapter 6, "Enhancing Vuser Scripts." For a list of the available Oracle NCA Vuser functions, see "Using Oracle NCA Vuser Functions" on page 901. For details of these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

In the following segment, the user selected an item from a list (**nca_list_activate_item**), pressed a button (**nca_button_press**), retrieved a list value (**nca_lov_retrieve_items**), and performed a click in an edit field (**nca_edit_click**). The logical names of the objects are the parameters of these functions.

```
…
nca_lov_select_item("Responsibilities","General Ledger, Vision Operations");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","+ Journals");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","       Enter");
nca_button_press("GLXJEENT.TOOLBAR.LIST.0");
nca_lov_find_value("Batches","");
nca_lov_retrieve_items("Batches",1,9);
nca_lov_select_item("Batches","AR 1020 Receivables 2537: A 1020");
nca_edit_click("GLXJEENT.FOLDER_QF.BATCH_NAME.0");
…
```

In certain tests, such as those performed on Oracle Configurator applications, information returned by one function is required throughout the session. VuGen automatically saves the dynamic information to a parameter, by inserting a **web_reg_save_param** function into the script. In the following example, the connection information is saved to a parameter called NCAJServSessionID.

```
web_reg_save_param ("NCAJServSessionId", "LB=\r\n\r\n", "RB=\r",
        LAST);
web_url("f60servlet",
        "URL=http://usscifforms05.sfb.na/servlet/f60servlet\?config
        =mult", LAST);
```

In the above example, the right boundary is \r. The actual right boundary may differ between systems.

Testing Oracle NCA Applications

The following sections contain several tips for testing secure Oracle NCA applications and servlets.

Testing Secure Oracle NCA Applications

- When selecting the protocols to record, you only need to select **Oracle NCA**—not **Web Protocol** from the protocol list. VuGen records the security information internally and therefore does not need the explicit Web functions.
- In the Port Mapping recording options, delete any existing entries for port 443 and create a new entry for the Oracle server name:

  Service ID: HTTP
  Target Server: Oracle Forms Server IP address or long host name
  Target Port: 443
  Connection Type: SSL
  SSL Version: Active SSL version. If in doubt, select SSL 2/3.

  For more information, see Chapter 78, "Configuring the Port Mappings."
- If you encounter problems when replaying an NCA HTTPS script during the **nca_connect_server** command, insert the following function at the beginning of the script.

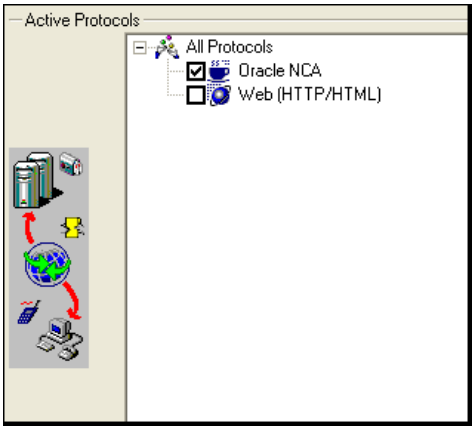
```
web_set_sockets_option("SSL_VERSION","3");
```

Testing Servlets and other Oracle NCA Applications

Certain NCA sessions use servlets:

- the Forms Listener servlet
- applications or modules that use both NCA and HTTP communications, such as the Oracle Configurator
- the initializing of the NCA application (downloading the applet, jar, and gif files)

When recording servlets, you must record both Oracle NCA and Web functions. You can do this by initially creating a multi-protocol script. Alternatively, if you created a single protocol script for Oracle NCA, open the **General:Protocols** node in the Recording Options, and enable the Web protocol. Then you can begin recording.



If you are unsure whether your application uses servlets, check the **default.cfg** file in the script directory. Locate the entry

UseServletMode=

If the value is 1 or 2, then servlets are being used and you must enable HTTP recording in addition to Oracle NCA.

If you already recorded a script, you can regenerate the code automatically to include the Web functions without having to re-record. Select **Tools > Regenerate Script**, and select the Web protocol in the Protocols section.

Determining the Recording Mode

When recording Oracle NCA scripts: VuGen automatically determines the correct connection mode: HTTP or Socket mode. Generally, you are not required to modify any of the recording settings as VuGen auto-detects the system configuration. In systems where the standard port mapping are reserved by other applications, you may need to modify the Port Mapping settings, depending on the recording mode.

You can determine the recording mode in one of the following ways:

➤ When using the NCA application, open the Java Console.

```
proxyHost=null
proxyPort=0
connectMode=HTTP
Forms Applet version is: 60812
```

The **connectMode** entry indicates **HTTP**, **HTTPS**, or **socket**.

➤ After recording an NCA session, open the **default.cfg** file in the Vuser directory and check the value of the **UseHttpConnectMode** entry.

```
[HttpConnectMode]
UseHttpConnectMode= 2
// 0 = socket 1 = http 2 = https
```

When defining a new port mapping int he Server Entry dialog box, use a **Service ID** of HTTP for HTTP or HTTPS modes. For Socket mode, use a **Service ID** of NCA.

For more information about Port Mapping settings, see Chapter 78, "Configuring the Port Mappings."

Recording Trace Information for Oracle DB

To debug your script, you can use the Oracle DB breakdown graphs. To gather data for this graph, you turn on the trace mechanism before running the script.

To manually turn on the tracing mechanism, use the **nca_set_custom_dbtrace** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Correlating Oracle NCA Statements for Load Balancing

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

To correlate statements for load balancing:

1 Record a multi-protocol script.

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

2 Define parameters for host and host arguments.

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

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

3 Assign values to serverHost and serverArgs:

```
web_url("step_name", "URL=http://server1.acme.com/test.htm", LAST);
```

4 Modify the nca_connect_server statement from:

```
nca_connect_server("199.203.78.170",
                   9000"/*version=107*/,
"module=e:\\appsnca…fndnam=apps ");
```

to:

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

The script should now look like this:

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

Additional Recommended Correlations

When recording an Oracle NCA session, VuGen records dynamic values—values that change for each record and replay session. Two common dynamic arguments are icx_ticket and JServSessionIdroot.

icx_ticket

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

```
web_url("fnd_icx_launch.runforms",
"URL=http://ABC-
123:8002/pls/VIS/fnd_icx_launch.runforms\?ICX_TICKET=5843A55058947ED3&RES
P_APP=AR&RESP_KEY=RECEIVABLES_MANAGER&SECGRP_KEY=STANDARD",
LAST);
```

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

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

…

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

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

JServSessionIdroot

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

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

```
vuser_init.c(8):    Set-Cookie: JServSessionIdroot=my1sanw2n1.JS4; path=/\r\n
vuser_init.c(8):    Content-Length: 79\r\n
vuser_init.c(8):    Content-Type: text/plain\r\n
vuser_init.c(8):    \r\n
vuser_init.c(8): 81-byte response body for "http://ABC-
123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo&ifhost=mercury&ifip=12
3.45.789.12" (RelFrameId=1)
vuser_init.c(8):
/servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdroot=my1sanw2n1.JS4\r\n
```

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

```
web_reg_save_param("NCAJServSessionId","LB=\r\n\r\n","RB=\r","ORD=1",LAST)
;

web_url("f60servlet",
    "URL= http://ABC-"123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo&"
"ifhost=mercury&ifip=123.45.789.12", LAST);

web_url("oracle.forms.servlet.ListenerSer",
    "URL=http://ABC-123{NCAJServSessionId}?ifcmd=getinfo&"
"ifhost=mercury&ifip=123.45.789.12", LAST);
```

Recording in Pragma Mode

The client side of the Oracle NCA Vuser can be configured to send an additional header to the server named **Pragma**. The header is a counter that behaves in the following way: the initial message of the NCA handshake has a value of 1.

The messages that follow the handshake are counted, beginning with 3. The counter is incremented by 1 for each message sent by the client.

If the message received from the server is the type plain\text and the body of the message begins with ifError:#/#00, the client sends a 0 byte message to the server and the Pragma value changes its sign to a minus. This sign changes back after the client succeeds in receiving the information from the server.

Recording of the Pragma header is only supported in the multi-protocol mode (Oracle NCA and Web). You can identify the Pragma mode within the script's default.cfg file. When operating in Pragma mode, the UseServletMode is set to 2.

```
[HttpConnectMode]
UseHttpConnectMode=1
RelativeURL=<NCAJServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see "Oracle NCA Run-Time Settings" on page 1318.

To identify the Pragma mode, you can perform a WinSocket level recording and check the buffer contents. In the first example, the buffer contains the Pragma values as a counter:

```
send buf108
    "POST /ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
    "vlet=gk5q79uqy1 HTTP/1.1\r\n"
    "Pragma: 1\r\n"
    ...
send buf110
    "POST /ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
    "vlet=gk5q79uqy1 HTTP/1.1\r\n"
    "Pragma: 3\r\n"
    ...
```

In the following example, the buffer contains the Pragma values as an error indicator:

```
recv buf129 281
     "HTTP/1.1 200 OK\r\n"
     "Date: Tue, 21 May 2002 00:03:48 GMT\r\n"
     "Server: Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_fastcgi/2.2"
     ".10 mod_perl/1.25 mod_oprocmgr/1.0\r\n"
     "Content-Length: 13\r\n"
     "Content-Type: text/plain\r\n"
     "\r\n"
     "ifError:8/100"

send buf130
     "POST /ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
     "vlet=gk5q79uqy1 HTTP/1.1\r\n"
     "Pragma: -12\r\n"
     ...
```

57

SAPGUI Protocol

In the growing field of ERP (Enterprise Resource Planning), SAP provides solutions allowing companies to manage all of their business processes. HP provides tools for testing SAP solution modules on both functional and load testing levels. This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). For information on testing solutions for mySAP Workplace and Portal clients, see Chapter 60, "SAP-Web Protocol."

This chapter includes:

About Developing SAPGUI Vuser Scripts

This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). To test the SAPGUI user operating only on the client, use the SAPGUI Vuser type. To test a SAPGUI user that also uses a Web browser, use the SAP (Click and Script) protocol.

Before recording a session, verify that your modules and client interfaces are supported by VuGen. The following table describes the SAP client modules for SAP Business applications and the relevant tools:

SAP module	VuGen support
SAP Web Client or mySAP.com.	Use the SAP-Web Vuser type.
SAPGUI for Windows.	A Windows-based client, emulated by the SAPGUI Vuser. This also supports APO module recording (requires patch level 24 for APO 3.0).
SAPGUI for Windows and a web browser.	Use the SAP (Click and Script) protocol.
SAPGUI for Java.	This client is not supported.

Version 6.20 and later:

- **For Functional Testing.** Use the QuickTest Professional Add-in for mySAP.com client.
- **For Load Testing.** Use the SAPGUI or SAP (Click and Script) protocol to create a script in VuGen and run a scenario in the Controller.

You use VuGen to record typical business processes. VuGen records SAPGUI for Windows client activity during SAP business processes, and generates a Vuser script. When you perform actions within the SAPGUI for Windows client, VuGen generates functions that describe this activity. Each function begins with a **sapgui** prefix.

Checking your Environment for SAPGUI Vusers

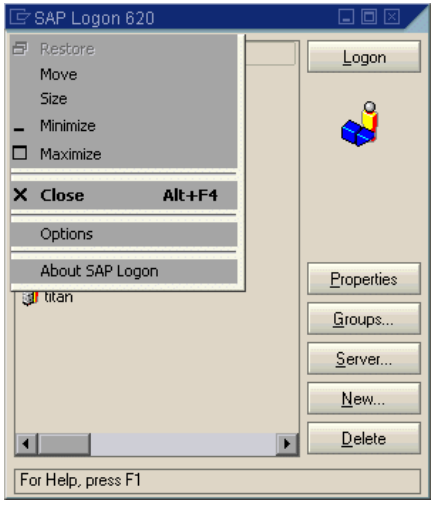
The basic steps in checking and setting up your system for the recording of SAPGUI Vusers, are Checking the Patch Level and Enabling Scripting. Once your environment is configured properly, you can record a typical SAP session and replay it in VuGen.

Checking the Patch Level

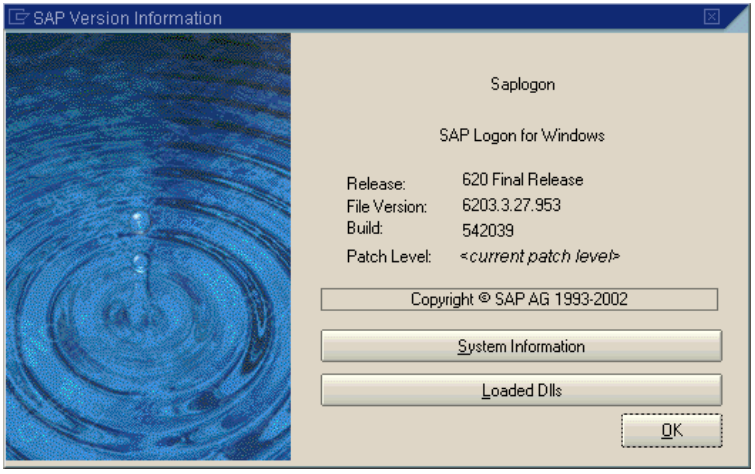
You can check the patch level of your SAPGUI for Windows client from the About box. The lowest patch level supported is version 6.20 patch 32.

To check the patch level:

1 Invoke the SAPGUI logon window. Click the top left corner of the SAP Logon dialog box and select **About SAP Logon** from the menu.



2 The SAP version information dialog box opens. Verify that the Patch Level entry is 32 or higher.



Enabling Scripting

VuGen support for the SAPGUI for Windows client, is based on SAP's Scripting API. This API allows Vusers to interact with the SAPGUI client, receive notifications, and perform operations.

The Scripting API is only available in recent versions of the SAP Kernel. In kernel versions that support scripting, the option is disabled by default. In order to use VuGen, first make sure that the SAP servers support the Scripting API, and enable the API on both the server and clients. For more information and to download patches, see the SAP OSS note #480149.

VuGen provides a utility that checks if your system supports scripting. The utility, **VerifyScript.exe**, is located on DVD in the **Additional Components\SAP_Tools\VerifySAPGUI** folder. For more information, see the file **VerifyScripting.htm** provided with this utility.

The following sections describe how to enable scripting.

- Checking the Configuration
- Enabling Scripting on the SAP Application Server
- Enabling Scripting on SAPGUI 6.20 Client

Checking the Configuration

The first step in enabling scripting is ensuring that the right kernel version is installed, and updating it if required.

Check the table below, for the minimum kernel patch level required for your version of the SAP Application Server. If required, download and install the latest patch.

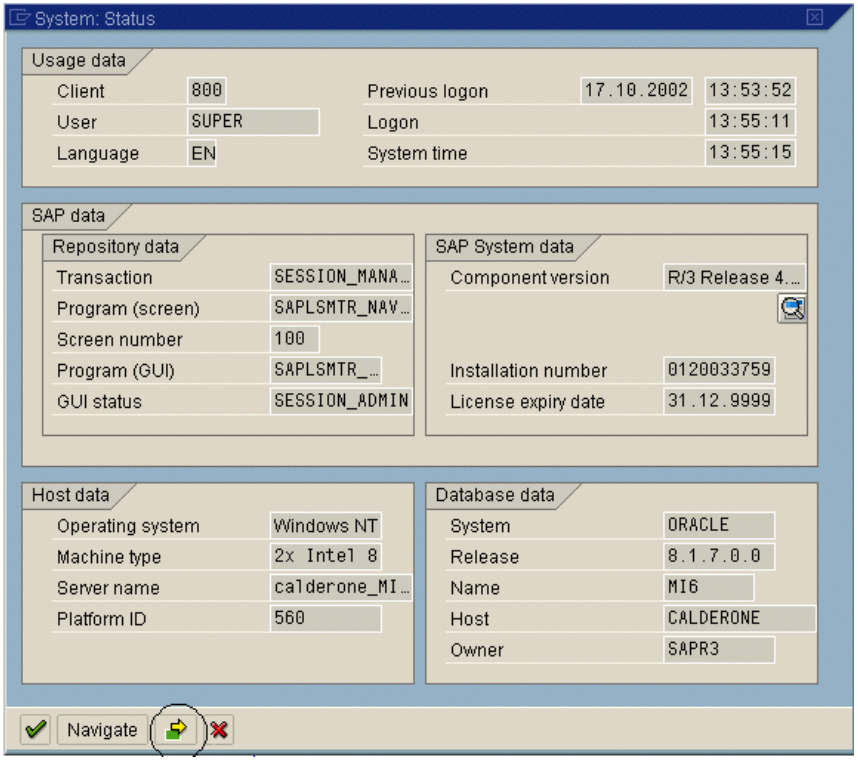
Software Component	Release	Package Name	Kernel Patch Level
SAP_APPL	31I	SAPKH31I96	Kernel 3.1I level 650
SAP_APPL	40B	SAPKH40B71	Kernel 4.0B level 903
SAP_APPL	45B	SAPKH45B49	Kernel 4.5B level 753
SAP_BASIS	46B	SAPKB46B37	Kernel 4.6D level 948
SAP_BASIS	46C	SAPKB46C29	Kernel 4.6D level 948
SAP_BASIS	46D	SAPKB46D17	Kernel 4.6D level 948
SAP_BASIS	610	SAPKB61012	Kernel 6.10 level 360

To check the kernel patch level:

1 Log in to the SAP system

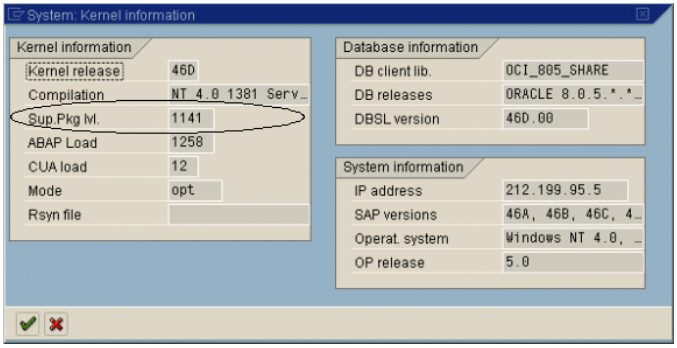
2 Select **System** > **Status**

3 Click the **Other kernel information button** (with the yellow arrow).



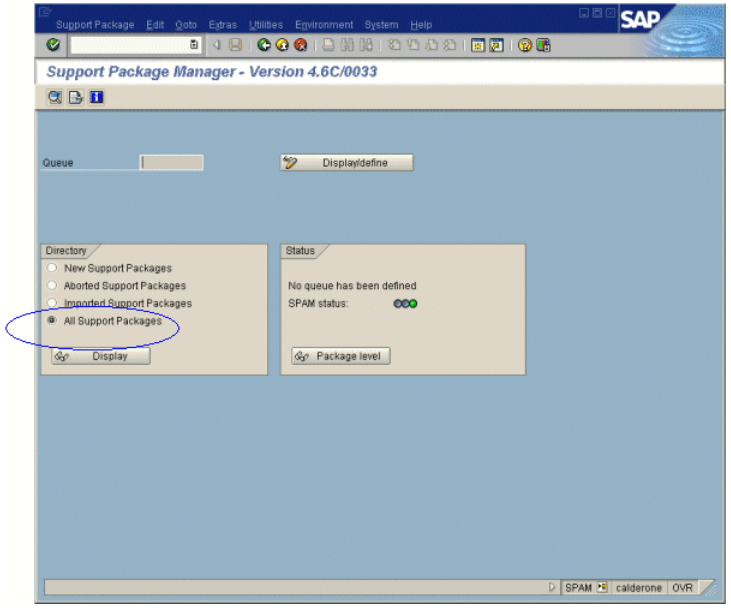
4 In the **Kernel Information** section, check the value of the **Sup. Pkg. lvl.**

If the level is lower than 948, you must download the latest kernel version and upgrade your existing one. See the SAP OSS note #480149 for detailed instructions on how to perform this upgrade.

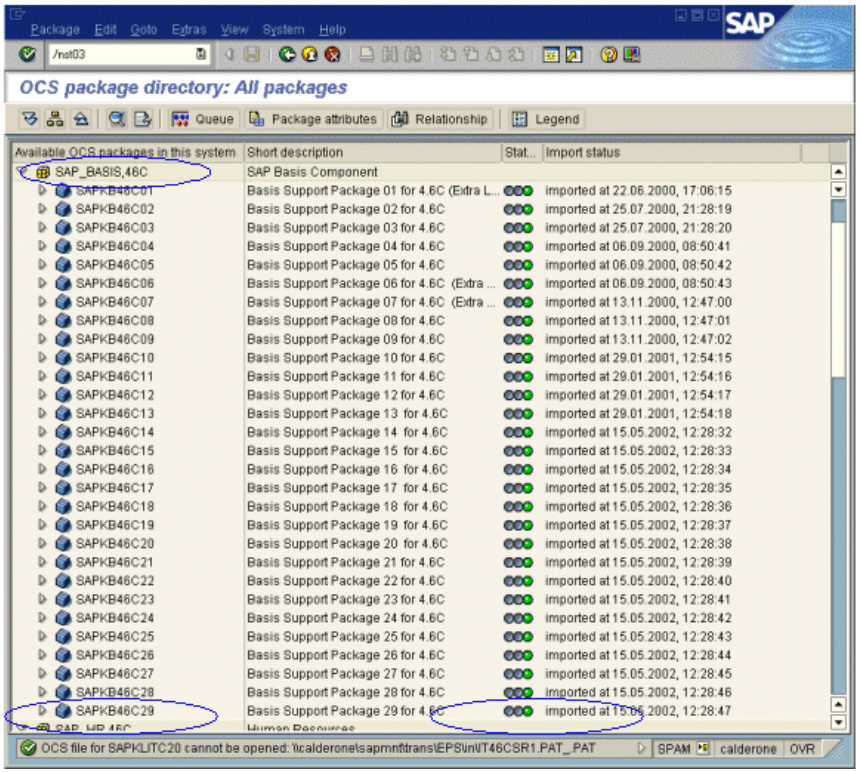


To check the R/3 support packages:

1. Log on to the SAP system and run the SPAM transaction.
2. In the **Directory** section, select **All Support Packages**, and click the **Display** button.



3 Verify that SAPKB46C29 is installed for SAP_BASIS, 4.6C. If it is installed, a green circle appears in the Status column.



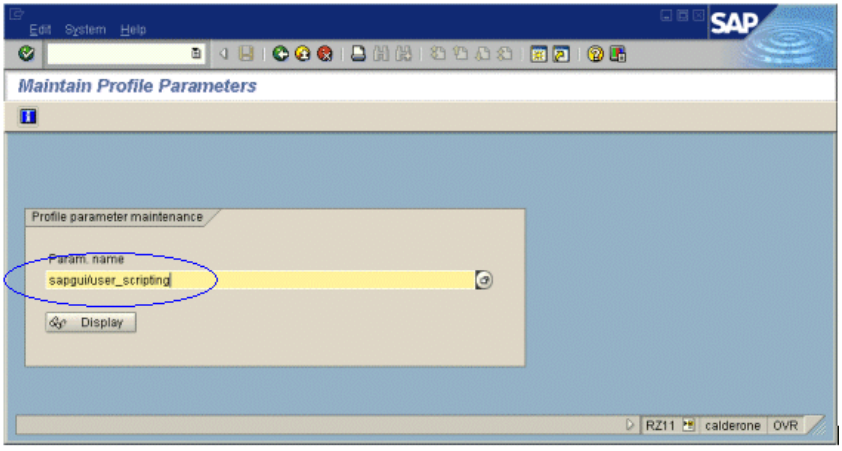
If you do not have the OCS package installed, download it the from the www.sap.com Web site and install it. For more information, see the SAP OSS note #480149.

Enabling Scripting on the SAP Application Server

A user with administrative permissions enables scripting by setting the **sapgui/user_scripting** profile parameter to **TRUE** on the application server. To enable scripting for all users, set this parameter on all application servers. To enable scripting for a specific group of users, only set the parameter on application servers with the desired access restrictions.

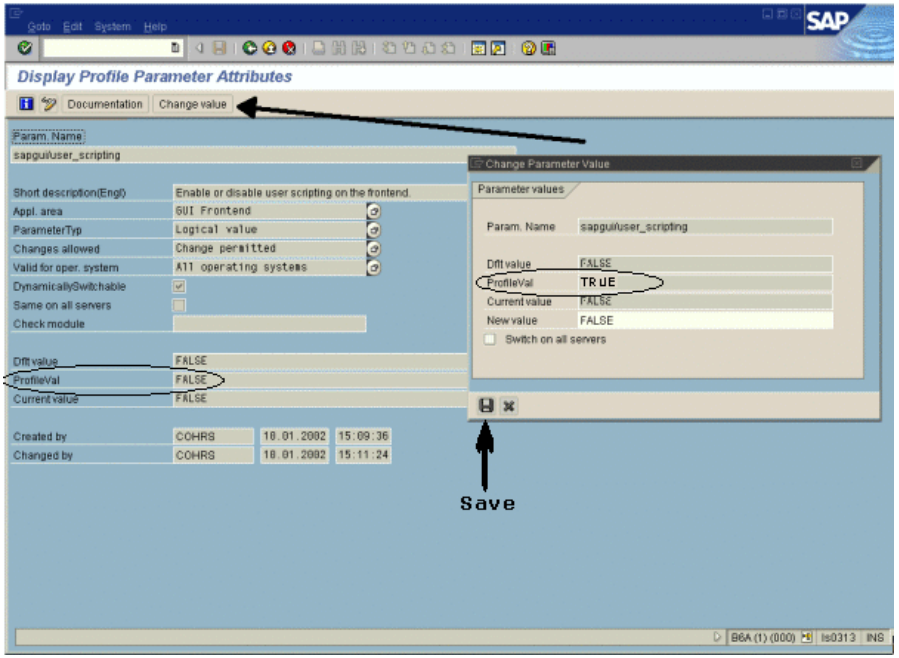
To change the profile parameter:

1 Open transaction **rz11.** Specify the parameter name **sapgui/user_scripting** and click **Display**. The Display Profile Parameter Attributes window opens.



If **Parameter name is unknown** appears in the status bar, this indicates that you are missing the current Support Package. Import the Support Package that corresponds to the SAP BASIS and kernel versions of the application server, as described in "Checking the Configuration" on page 917.

2 If **Profile Val** is FALSE, you need to modify its value. Click the **Change value** button in the toolbar. The Change Parameter Value window opens. Enter TRUE in the **ProfileVal** box and click the **Save** button.



When you save the change, the window closes and **ProfileVal** is set to TRUE.

3 Restart the application server, since this change only takes effect when you log onto the system.

If the updated **ProfileVal** did not change, even after restarting the server, then the kernel of the application server is outdated. Import the required kernel patch, as specified in the section "Checking the Configuration" on page 917.

Note that the Profile Value may be dynamically activated in the following kernel versions, using transaction rz11, without having to restart the application server.

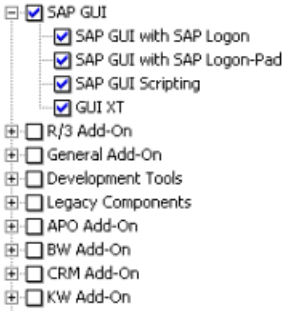
Release	Kernel Version	Patch Level
4.6B, 4.6C, 4.6D	4.6D	972
6.10	6.10	391
6.20	all versions	all levels

Enabling Scripting on SAPGUI 6.20 Client

To allow VuGen to run scripts, you must also enable scripting on the SAPGUI client. You should also configure the client not to display certain messages, such as when a connection is established, or when a script is attached to the GUI process.

To configure the SAPGUI client to work with VuGen:

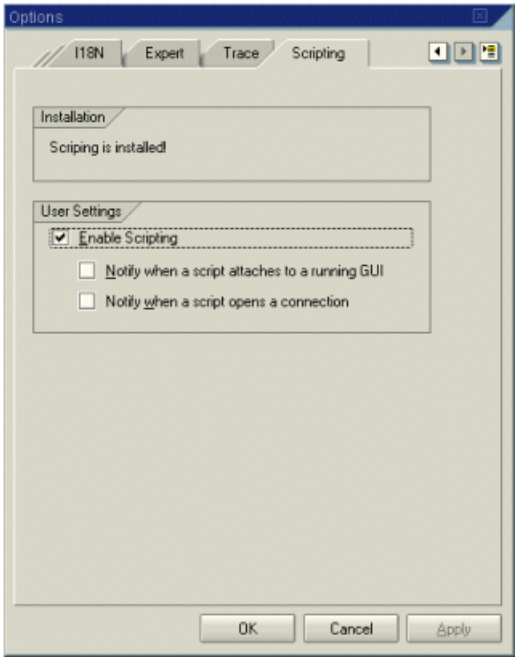
➤ **During installation.** While installing the SAPGUI client, enable the **SAP GUI Scripting** option.



➤ **After installation.** Suppress warning messages. Open the Options dialog box in the SAPGUI client. Select the **Scripting** tab and clear the following options:

1 Notify when a script attaches to a running GUI

2 Notify when a script opens a connection



You can also prevent these messages from popping up by setting the values **WarnOnAttach** and **WarnOnConnection** in the following registry key to 0:

HKCU\SOFTWARE\SAP\SAPGUI Front\SAP Frontend Server\Security.

Creating a SAPGUI Vuser Script

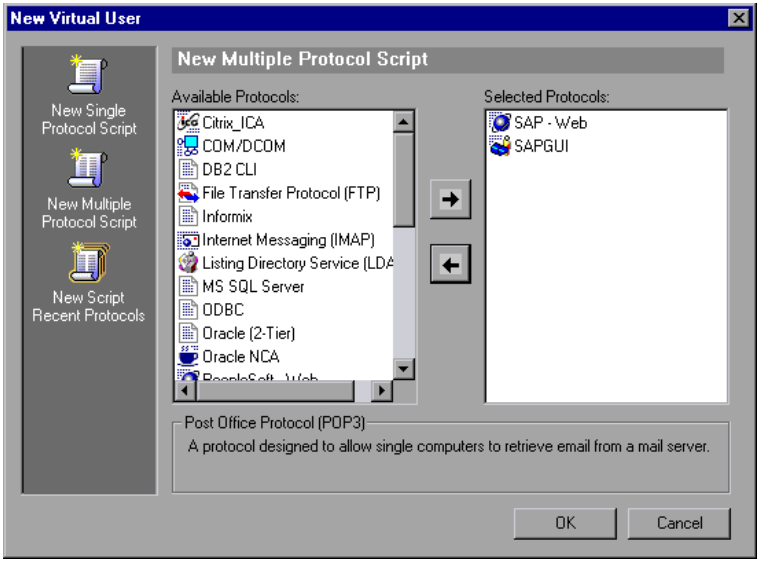
The first step in creating a SAPGUI Vuser script is choosing the Vuser and script type. The SAP Vuser type, **SAPGUI** is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script.

To create a SAPGUI Vuser script:

1 Invoke VuGen and select **File** > **New**.

2 To record a standard SAPGUI client session (with no browser controls), create a single-protocol Vuser script using the **SAPGUI** type Vuser.

3 To record a SAPGUI session that uses browser controls, create a multi-protocol Vuser script. Specify both the **SAPGUI** and **SAP-Web** Vuser types. This allows VuGen to record Web-specific functions when encountering the browser controls.



4 Click **OK** to open the Vuser script.

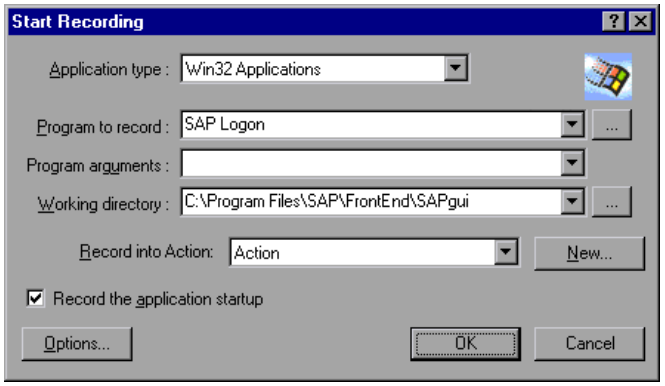
Recording a SAPGUI Vuser Script

After creating an empty script, you set the recording options and then record your SAPGUI session. VuGen generates a script corresponding to your actions within the client.

To begin recording a SAPGUI script:

1 If the Start Recording dialog box was not opened, click the **Start Recording** button. The Start Recording dialog box opens.

2 VuGen detects and fills in the relevant information:



- **Program to record.** VuGen locates the saplogon.exe file in the SAP client installation.
- **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.
- **Record into Action.** Select the section into which you want to record. Initially, the available sections are vuser_init, Action1, and vuser_end

3 Click **OK** and begin recording.

Recording at the Cursor

VuGen also allows you to record actions into an existing script. You may decide to record into an existing script for several reasons:

- You made a mistake in the actions that you performed during recording.
- Your actions were correct, but you need to add additional information such as the handling of popup windows. For example the SAP server may issue an inventory warning, which did not apply during the recording session.

This feature, called Recording at the Cursor, lets you insert new actions or replace existing actions. When you begin Recording at the Cursor, VuGen prompts you with two options:

- Insert steps into action. Inserts the newly recorded steps at the cursor without overwriting any existing steps. The new segment is enclosed with comments indicating the beginning and end of the added section. This option is ideal for handling occasional popup windows that were not present during the recording

```
// Recording at the cursor - Begin
    sapgui_select_active_connection("con[0]");
    sapgui_select_active_session("ses[0]");
    sapgui_select_active_window("wnd[0]");
//Recording at the cursor - End
```

- **Overwrite the rest of the script.** Replaces all steps from the point of the cursor onward. This option overwrites the remainder of the current Action and deletes all other Actions. It does not affect the **vuser_init** or **vuser_end** sections.

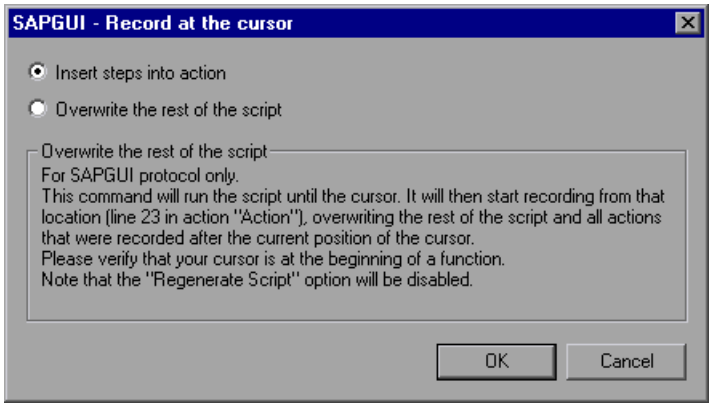
After you select one of the Recording at the Cursor options, VuGen replays the script from the beginning until the cursor location. Then it opens the Recording floating toolbar and begins recording. If you use the **Recording at the Cursor** feature, the **Regenerate Script** tool becomes disabled.

Note: To record at the cursor, you need to click in the left margin of the VuGen editor, immediately before an existing function.

To Record at the Cursor:

1 Open Script view (**View** > **Script view**) and click in the left margin adjacent to an existing function.

2 Click the **Recording at the Cursor** button. VuGen prompts you to make a selection.



3 Select **Insert steps into action** or **Overwrite the rest of the script**. Click **OK**. VuGen replays the script until the point of the cursor.

4 Wait for the Recording floating toolbar to open. Then begin performing actions in the SAPGUI client, switching between sections and actions as required.

5 Click the **Stop** button to end the recording session.

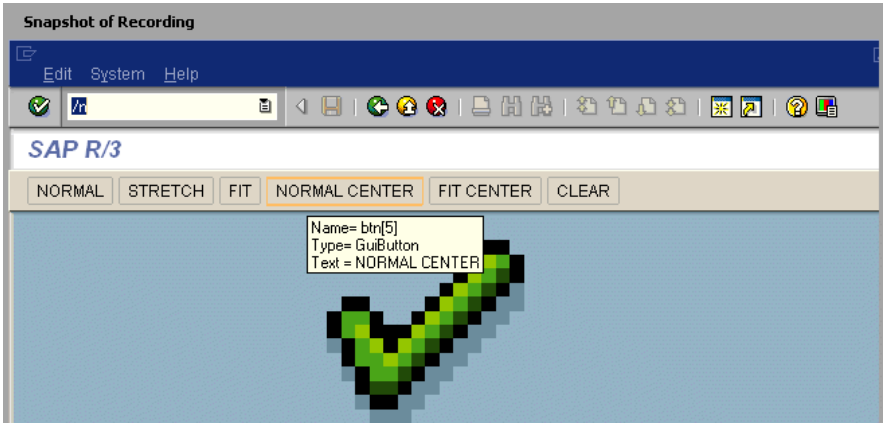
Inserting Steps Interactively into a SAPGUI Script

After recording, you can manually add steps to the script in either Script view and Tree View. For information about adding steps from the various views, see "Viewing and Modifying Vuser Scripts" on page 48.

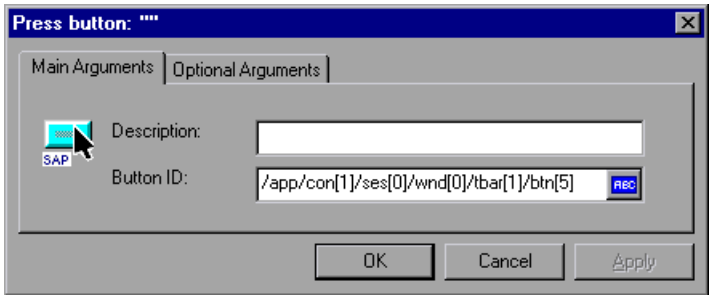
In addition to manually adding new functions, you can add new steps interactively for SAPGUI Vusers, directly from the snapshot. Using the right-click menu, you can add object-related steps.

When adding a step from within a snapshot, VuGen uses the Active Screen capability and determines the ID of each object in the SAPGUI client window (unless you disabled Active Screen snapshots in the SAPGUI General Recording Options).

To determine which objects were recognized by VuGen, you move the mouse over the snapshot. VuGen draws a box around the objects as you pass over them and displays a tool tip with the object's Control ID. In the following example, the selected active object is the NORMAL CENTER button.



When you add a step while holding the mouse over a recognized object, VuGen automatically inserts the Control ID of that object into the relevant field of the Properties dialog box. For example, if you add a **Press Button** step, for the NORMAL CENTER button as shown above, the Properties box displays the following ID:



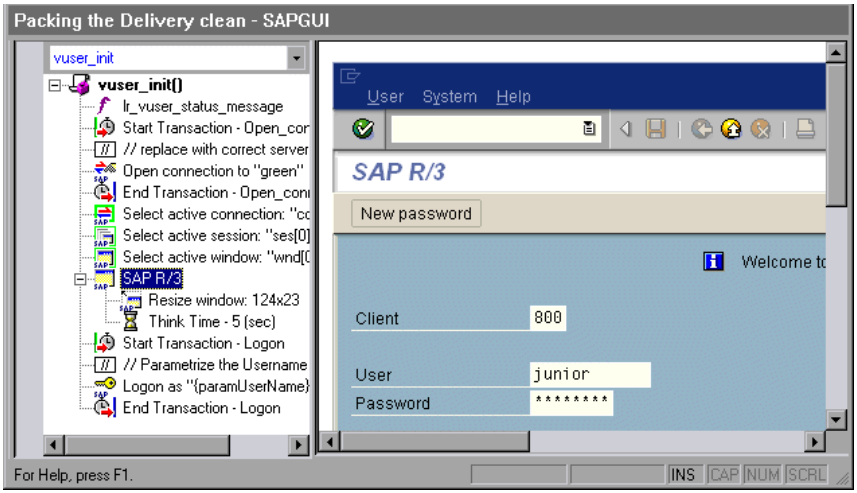
To insert a step interactively for a specific object:

1. Click within the Snapshot window.
2. Move the mouse over the object for which you want to add a function. Make sure that VuGen recognizes the object and encloses it with a box.
3. Select **Insert New Step** from the right-click menu. The Insert Step box opens.
4. Select a step from the menu. The step's Properties dialog box opens, with the Control ID of the object when relevant.
5. Enter a name for the object in the **Description** box. Click **OK**. VuGen inserts the new step after the selected step.
6. To get the Control ID of the object for the purpose of pasting it into a specific location, select Copy **Control ID** from the right-click menu. VuGen places it on the clipboard. You can past it into a Properties box or directly into the code from the Script view.

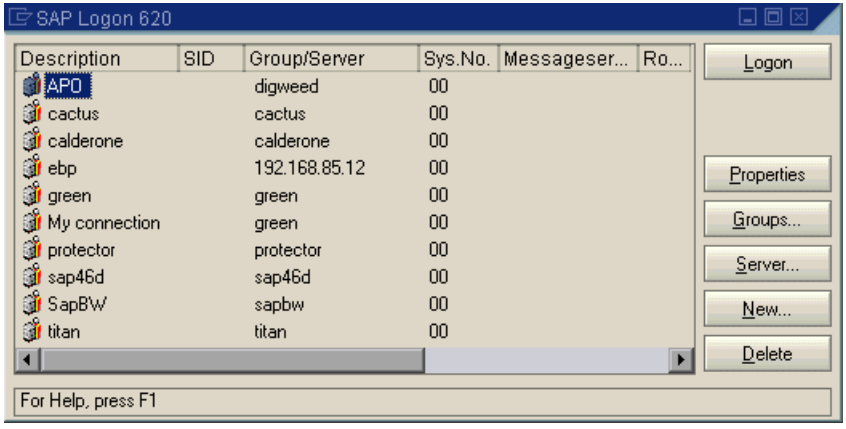
Understanding a SAPGUI Vuser Script

The SAPGUI Vuser script typically contains several SAP transactions which make up a business process. A business process consists of functions that emulate user actions. Open the tree view to see each user action as a Vuser script step.

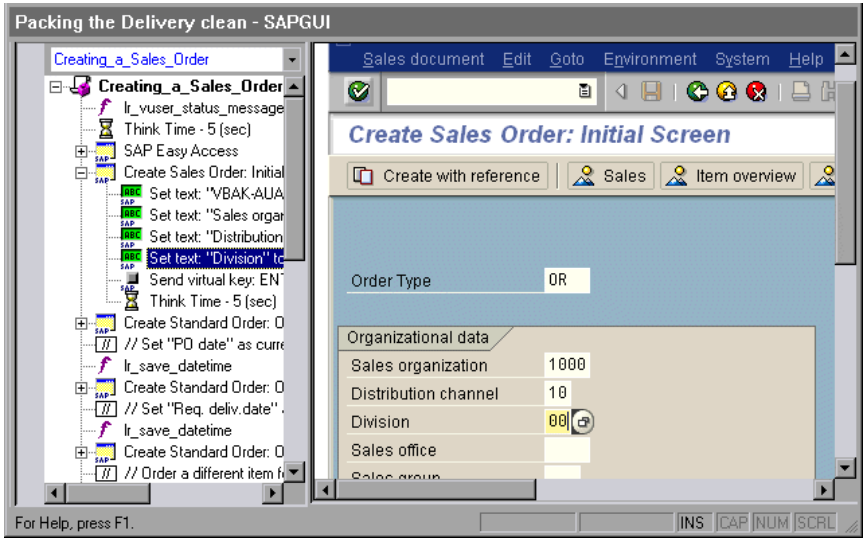
The following example shows a typical recording of a SAPGUI client. The first section, **vuser_init**, contains the opening of a connection and a logon.



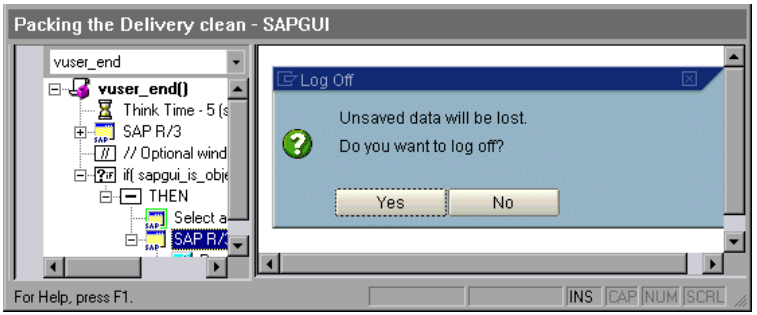
Note that the Open Connection step uses one of the connection names in the SAP Logon **Descriptions** list. If the specified connection name is not in the list, the Vuser looks for a server with that name.



In the following section, the functions emulate typical user operations such as menu selection and the setting of a check box.



The final section, **vuser_end**, illustrates the logoff procedure.



When recording a multi- protocol script for both SAPGUI and Web, VuGen generates steps for both protocols. In the Script view, you can view both **sapgui** and **web** functions.

The following example illustrates a multi-protocol recording in which the SAPGUI client opens a Web control. Note the switch from **sapgui** to **web** functions.

```
sapgui_tree_double_click_item("Use as general WWW browser, REPTITLE",
    "shellcont/shell",
    "000732",
    "REPTITLE",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1020",
    END_OPTIONAL);

...
sapgui_set_text("",
    "http:\\\\yahoo.com",
    "usr/txtEDURL",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1021",
    END_OPTIONAL);

...
web_add_cookie("B=7pt5cisv1p3m2&b=2; DOMAIN=www.yahoo.com");

web_url("yahoo.com",
        "URL=http://yahoo.com/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        EXTRARES,
"URL=http://srd.yahoo.com/hpt1/ni=17/ct=lan/sss=1043752588/t1=1043752575385/d1
=1251/d2=1312/d3=1642/d4=4757/0.4097009487287739/*1",
"Referer=http://www.yahoo.com/", ENDITEM,
        LAST);
```

Enhancing a SAPGUI Vuser Script

After you examine the recorded Vuser script, you enhance it in the following ways:

- **Transactions.** Inserting transactions, rendezvous points, and control-flow structures into the script. For details, see Chapter 6, "Enhancing Vuser Scripts."
- **Verification.** Insert SAPGUI verification functions to verify the current state of SAPGUI objects. For details, see Adding Verification Functions.
- **Retrieve information.** Insert SAPGUI functions to verify the current value of SAPGUI objects. You use the **sapgui_get_xxx** functions to retrieve information. For more information, see "Retrieving Information" on page 937.

  Define parameters (optional). Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values. For details, see Chapter 70, "Working with VuGen Parameters."

Adding Verification Functions

When working with optional or dynamic windows or frames, you can use verification functions to determine if the window or object is available. An optional window is a window that does not consistently open during the SAP session. This function allow the Vuser script to continue running even if an optional window opens or an exception occurs.

The first example checks if a window is available. If the window is available, the Vuser closes it before continuing.

```
if (!sapgui_is_object_available("wnd[1]"))
        sapgui_call_method("{ButtonID}",
            "press",
            LAST,
            AdditionalInfo=info1011");
sapgui_press_button(.....)
```

The next example illustrates a dynamic object in the ME51N transaction. The Document overview frame is optional, and can be opened/closed by the **Document overview on/off** button.

The code checks the text on the Document overview button. If the text on the button shows Document overview on, we click the button to close the Document overview frame.

```
if(sapgui_is_object_available("tbar[1]/btn[9]"))
{
    sapgui_get_text("Document overview on/off button",
        "tbar[1]/btn[9]",
        "paramButtonText",
        LAST);

    if(0 == strcmp("Document overview off", lr_eval_string("{paramButtonText}")))
        sapgui_press_button("Document overview off",
            "tbar[1]/btn[9]",
            BEGIN_OPTIONAL,
                "AdditionalInfo=sapgui1013",
            END_OPTIONAL);
}
```

Retrieving Information

When working with SAGUI Vusers, you can retrieve the current value of a SAPGUI object using the **sapgui_get_<xxx>** functions. You can use this value as input for another business process, or display it in the output log.

Retrieving Status Bar Information

The following example illustrates how to save part of a status bar message in order to retrieve the order number.

To retrieve the order number from the status bar:

1. Navigate to the point where you want to check the status bar text, and select **Insert > New Step.** Select the **sapgui_status_bar_get_type** function. This verifies that the Vuser can successfully retrieve text from the status bar.
2. Insert an **if** statement that checks if the previous statement succeeded. If so, save the value of the argument using **sapgui_status_bar_get_param**.

This sapgui_status_bar_get_param function saves the order number into a user-defined parameter. In this case, the order number is the second index of the status bar string.

```
sapgui_press_button("Save (Ctrl+S)",
    "tbar[0]/btn[11]",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1038",
    END_OPTIONAL);

sapgui_status_bar_get_type("Status");
if(0==strcmp(lr_eval_string("{Status}"),"Success"))
  sapgui_status_bar_get_param("2", " Order_Number ");
```

During test execution, the Execution log indicates the value and parameter name:

```
Action.c(240): Pressed button " Save (Ctrl+S)"
Action.c(248): The type of the status bar is "Success"
Action.c(251): The value of parameter 2 in the status bar is "33232"
```

Saving Date Information

When creating scripts that use dates, your script may not run properly. For example, if you record the script on June 2, and replay it on June 3, the date fields will be incorrect. Therefore, you need to save the date to a parameter during text execution, and use the stored value as input for other date fields. To save the current date or time during script execution, use the **lr_save_datetime** function. Insert this function before the function requiring the date information. Note that the format of the date is specific to your locale. Use the relevant format within the **lr_save_datetime** function. For example, for month.day.year, specify "%m.%d.%Y".

In the following example, lr_save_datetime saves the current date. The **sapgui_set_text** function uses this value to set the delivery date for two days later.

```
lr_save_datetime("%d.%m.%Y", DATE_NOW + (2 * ONE_DAY),
    "paramDateTodayPlus2");

sapgui_set_text("Req. deliv.date",
        "{paramDateTodayPlus2}",
        "usr/ctxtRV45A-KETDAT",
        BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui1025",
        END_OPTIONAL);
```

58

SAPGUI - Replaying Scripts

After creating a SAPGUI script through recording and manual enhancements, you replay it in VuGen to test its functionality.

This chapter includes:

The following information applies to the SAPGUI and the SAP (Click and Script) protocols.

About Replaying SAPGUI Vuser Scripts

This chapter discusses the running of a SAPGUI Vuser script. You can set run-time settings to control the Vuser's behavior during the test or monitoring session.

This chapter also contains several guidelines for working with SAPGUI Vusers, as well as a troubleshooting section for solving common issues.

The SAPGUI Vuser script emulates a typical business processes using SAPGUI functions that begin with a **sapgui** prefix.

During replay, these functions emulate user activity on SAPGUI objects.

For example, sapgui_select_radio_button selects the radio button Blue.

```
sapgui_select_radio_button("Blue",
      "usr/radRB7",
      BEGIN_OPTIONAL,
         "AdditionalInfo=sapgui1027",
      END_OPTIONAL);
```

Replaying SAPGUI Optional Windows

When working with SAPGUI Vuser Scripts, you may encounter optional windows in the SAPGUI client—windows that were present during recording, but do not exist during replay. If you try to replay your recorded script as is, it will fail when it attempts to find the missing windows.

VuGen's optional window mechanism performs the actions on a window only after verifying that it exists. The Vuser checks if the window indicated in the **Select active window** step exists. If the window is found during replay, it performs the actions as they were recorded in the script. If it does not exist, the Vuser ignores all window actions until the next **Select active window** step. Note that only SAPGUI steps (beginning with a **sapgui** prefix) are ignored.

To use this feature, in Tree view select the appropriate Select Active Window step and select **Run steps for window only if it exists** from the right-click menu.

To disable this feature and attempt to run these steps at all times, regardless of whether the Vuser finds the window or not, select **Always run steps for this window** from the right-click menu.

SAPGUI Functions

During a SAPGUI recording session, VuGen generates functions that emulate user interaction with the SAPGUI client. When you record the SAPGUI for Windows client, VuGen generates functions with a **sapgui** prefix. This section lists all of the **sapgui** functions.

When you record a SAP session using a Web interface such as SAP Workplace or Portal, or if the SAPGUI client opens a Web control, VuGen generates functions with a **web** prefix.

While most of the functions are recorded, you can manually insert any function into your script. The functions that are not recorded are the data retrieval functions beginning with **sapgui_get**, and those used for verification, beginning with **sapgui_is**.

There are several categories of **sapgui** functions: Connection and Session Functions, Method and Property Functions, Verification and Data Retrieval Functions, and Object functions. Object functions are those which perform an action within a SAPGUI object such as Calendar Functions, Grid Functions, APO Grid Functions, Status Bar Functions, Table Functions, Tree Functions, Window Functions, and General Object Functions.

For more information about the **sapgui** and **web** functions, use the **Show Function Syntax** feature from the Edit menu, or see the *Online Function Reference* (**Help** > **Function Reference**).

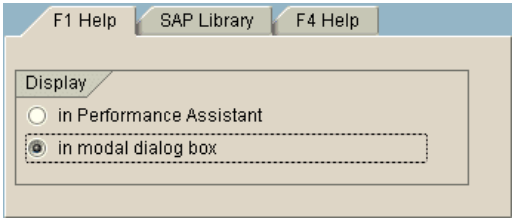
Tips for SAPGUI Vuser Scripts

The following sections provides Recording Tips, Replay Tips, and Tips for Replaying in a Scenario for SAPGUI Vusers. In addition, you can obtain information directly from the SAP support site.

Recording Tips

This section provides recording tips for a SAPGUI Vuser script.

- Make sure to record the actions into the appropriate sections: Record the logon procedure into the **vuser_init** section, the actions that you want to repeat in the **Actions** sections, and the logoff procedure in the **vuser_end** section.
- When recording a multi-protocol script in which the SAPGUI client contains Web controls, close the SAPLogon application before recording.
- Use modal dialog boxes for F1. Instruct the SAPGUI client to open the F1 help in a modal dialog box. Select **Help >Settings**. Click the **F1 Help** tab and select the **in modal dialog box** option in the Display section.



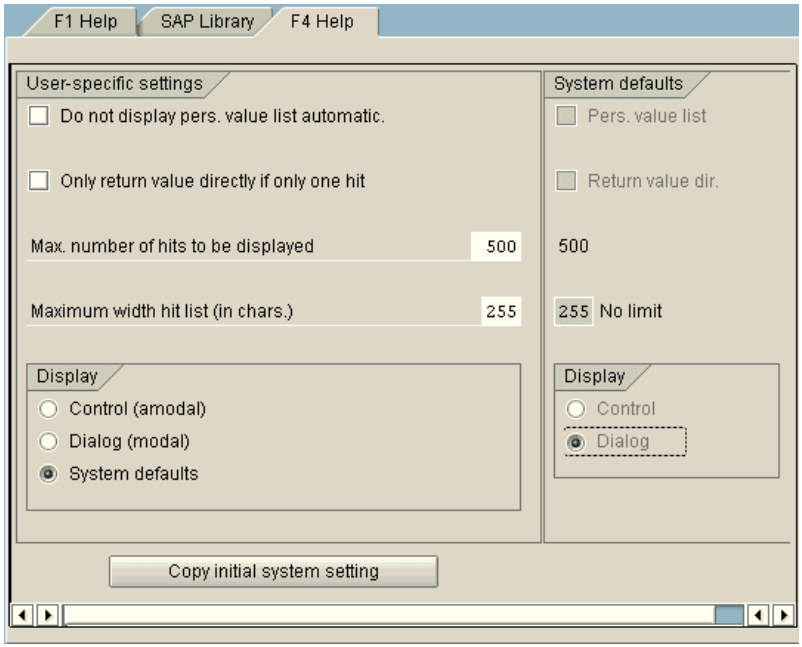
- Use modal dialog boxes for F4. Instruct the SAPGUI client to open the F4 help in a modal dialog box.

The following procedure must be performed by a SAP administrator:

To open F4 help in modal dialog boxes:

1 Make sure that all users have logged off from the server.

2 Select **Help** > **Settings**. Click the **F4 Help** tab.



3 In the Display section (bottom left), select System defaults.

4 In the Display portion of the System defaults section (bottom right), select **Dialog**.

5 Save the changes—click **Copy initial system setting** or CTRL+S.

6 Verify that the status bar displays the message **Data was saved**.

7 Close the session.

8 Restart the service through the SAP Management Console.

Replay Tips

Follow these guidelines before replaying your script in standalone-mode:

- Replace the encrypted password in the **sapgui_logon** function generated during recording, with the real password. It is the second argument of the function, after the user name: `sapgui_logon("user", "pswd", "800", "EN");` For additional security, you can encrypt the password within the code. Select the password text (the actual text, not *****) and select **Encrypt string** from the right-click menu. VuGen inserts an **lr_decrypt** function at the location of the password: `sapgui_logon("user", lr_decrypt("3ea037b758"), "800", "EN");`.
- When running a script for the first time, configure VuGen to show the SAPGUI user interface during replay, in order to see the operations being performed through the UI. To show the user interface during replay, open the run-time settings (F4) and select the **Show SAP Client During Replay** option in the **SAPGUI:General** node. During a load scenario, disable this option, since it uses a large amount of system resources in displaying the UI for multiple Vusers.

Tips for Replaying in a Scenario

The following sections provide configuration tips for running the script on a Controller or Load Generator machine.

Controller Settings

When working with a LoadRunner scenario, set the following values when running your script in a load test configuration:

- **Ramp-up.** One by one (to insure proper logon) in the Scheduler.
- **Think time.** Random think time in the Run-Time settings.
- **Users per load generator.** 50 Vusers for machine with 512 MB of memory in the Load Generators dialog box.

Load Generator Settings

When running your script in a scenario, check the agent mode and configure the terminal sessions on the Load Generator machines.

➤ **Agent Mode.** Make sure that the LoadRunner (or Performance Center) Remote Agent is running in Process mode. Service mode is not supported.

To check this, move your mouse over the agent's icon in the Windows task bar area, and read the description. If the description reads LoadRunner Agent Service, it is running as a service.

To restart the agent as a process:

1 Stop the agent. Right-click the LoadRunner Agent icon and select **Close**.

2 Run **magentproc.exe**, located in the **launch_service\bin** directory, under the LoadRunner installation.

3 To make sure that the correct Agent is launched the next time you start your machine, change the Start type of the Agent Service from Automatic to Manual. Then add a shortcut to **magentproc.exe** to the Windows Startup folder.

Terminal Sessions. Machines running SAPGUI Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine. To increase the number of Vusers per machine, open additional terminal server sessions on the Load Generator machines. Select **Agent Configuration** from **Start** > **Programs** > **<product_name>** > **Advanced Settings**, and select the **Enable Terminal Service** option. You specify the number of terminal sessions in the Load generator machine properties. For more information, see Configuring Terminal Services in the *HP LoadRunner Controller User's Guide*.

Note: When the LoadRunner Agent is running in a terminal session, and the terminal session's window is minimized, no snapshots will be captured on errors.

Troubleshooting SAPGUI Vuser Scripts

Question 1: I was able to record a script, but why does replay fail?

Answer: In LoadRunner, make sure that the LoadRunner Remote Agent is running in Process mode. Service mode is not supported. For more information, see "Replay Tips" on page 946.

Question 2: Why were certain SAPGUI controls not recorded?

Answer: Some SAPGUI controls are only supported in their menu or toolbar contexts. Try performing the problematic task using a different means—through a menu option, context menu, toolbar, and so on.

Question 3: Why can't I record or replay any scripts in VuGen?

Answer:

- **a** Verify that you have the latest patch of SAPGUI 6.20 installed. The lowest allowed patch level is patch 32.
- **b** Make sure that scripting is enabled. See the "Checking your Environment for SAPGUI Vusers" on page 915.
- **c** Verify that notifications are disabled in the SAPGUI for Windows client. Click the Customizing of Local Layout button or press ALT+F12. Click **Options** and select the Scripting tab. Clear both **Notify** options.

Question 4: What is the meaning of the error popup messages that are issued when I try to run the script?

Answer: Certain SAP applications store the last layout for each user (such as which frames are visible or hidden). If the stored layout was changed since the script was recorded, this may cause replay problems. For Example, in the ME52N transaction, the **Document overview Off/On** button will change the number of visible frames.

If this occurs:

1. Navigate the transaction to the same point as it was during recording, before starting replay. You can use the Snapshot viewer to see the layout in which it was recorded.
2. Add statements to the script that bring the transaction to the desired layout during replay. For example, if an optional frame interferes with your replay, insert a verification function that checks if the frame is open. If it is open, click a button to close it. For verification examples, see "Adding Verification Functions" on page 936.

Question 5: Can I use the single sign-on mechanism when running a script on a remote machine?

Answer: No, VuGen does not support the single sign-on connection mechanism. In your SAPGUI client, open the Advanced Options and clear the **Enable Secure Network Communication** feature. Note that you must re-record the script after you modify the Connection preferences.

Question 6: Can VuGen record all SAP objects?

Answer: Recording is not available for objects not supported by SAPGUI Scripting. See your recording log for information about those objects.

Question 7: Are all business processes supported?

Answer: VuGen does not support business processes that invoke Microsoft Office module controls, nor those that require the use of GuiXT. You can disable **GuiXT** from the SAPGUI for Windows client Options menu.

Additional Resources

LoadRunner

For Online Help on dialog boxes, press F1 within a dialog box. You can also select **Help** > **Contents and Index** to manually open the Help. In the Index tab, locate the **SAPGUI Vuser scripts** entry and click the appropriate sub-entry.

For Online Help with a function, click within the function or step, and click F1 to open the *Online Function Reference*.

SAP

For more information, see the SAP website at www. sap.com or one of the following locations:

➤ **SAP Notes** - https://websmp103.sap-ag.de/notes

Note #480149: New profile parameter for user scripting on the front end

Note #587202: Drag & Drop is a known limitation of the SAPGUI interface

➤ **SAP Patches** - https://websmp104.sap-ag.de/patches

SAP GUI for Windows - SAPGUI 6.20 Patch (the lowest allowed level is 32)

59

SAP (Click and Script) Protocol

VuGen allows you to create scripts that emulate SAP applications over the Web.

This chapter includes:

About Developing SAP (Click and Script) Vuser Scripts

VuGen can create test scripts for SAP Enterprise portal7 and SAP ITS 6.20/6.40 environments using specialized test objects and methods that have been customized for SAP. The objects are APIs based on HP QuickTest support for SAP.

As you record a test or component on your SAP application, VuGen records the operations you perform. VuGen recognizes special SAP Windows objects such as frames, table controls, iViews, and portals.

VuGen supports recording for the following SAP controls: button, checkbox, drop-down menu, edit field, iview, list, menu, navigation bar, OK code, portal, radio group, status bar, tab strip, table, and tree view.

Recording a SAP (Click and Script) Session

To create a Vuser script that emulates SAP Web applications, you select the **SAP (Click and Script)** protocol type from the **ERP** category. To begin recording, click the **Record** button and perform typical actions in your SAP Web application. You should record your sign-in information in the **vuser_init** section, and the sign off process in the **vuser_end** section. For further information about creating and recording a script, see Chapter 5, "Recording with VuGen."

You can set event related recording options. See Chapter 74, "Click and Script Recording" for more information.

If you require a lower level script, or if you need to record on unsupported SAP control, use the **SAP-Web** protocol in the ERP category.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding SAP (Click and Script) Scripts

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported SAP objects, VuGen generates a function with an **sap_xxx** prefix.

In the following example, a user selected the **User Profile** tab. VuGen generated a **sap_portal** function.

```
web_browser("Close_2",
        "Snapshot=t7.inf",
        DESCRIPTION,
        "Ordinal=2",
        ACTION,
        "UserAction=Close",
        LAST);

lr_think_time(7);

web_text_link("Personalize",
        "Snapshot=t8.inf",
        DESCRIPTION,
        "Text=Personalize",
        ACTION,
        "UserAction=Click",
        LAST);

    lr_think_time(6);

sap_portal("Sap Portal_2",
        "Snapshot=t9.inf",
        DESCRIPTION,
        "BrowserOrdinal=2",
        ACTION,
        "DetailedNavigation=User Profile",
        LAST);
```

Note: When you record a SAP (Click and Script) session, VuGen generates standard Web (Click and Script) functions for objects that are not SAP-specific. You do not need to explicitly specify the Web protocol. In the example above, VuGen generated a **web_text_link** function when the user clicked the **Personalize** button.

60

SAP-Web Protocol

You use VuGen's SAP-Web Vuser type, to record the activity in SAP Workplace or SAP Portal clients.

This chapter includes:

About Developing SAP-Web Vuser Scripts

You use VuGen to record typical SAP business processes. VuGen records SAP Workplace or Portal activity during the business processes, and generates a Vuser script. When you perform actions within your browser, VuGen generates functions that describe this activity. Each function begins with a **web** prefix.

During replay, these functions emulate user activity on the SAP Workplace or Portal clients. For example, web_url navigates to the PageBuilder.

```
web_url("PageBuilder[myPage]",
"URL=http://sonata.hplab.com/hrnp$30001/sonata.hplab.co.il:80/Action/PageBuilder[m
yPage]?pageName=com.sapportals.pct.home.mynews",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.hplab.co.il/sapportal",
        "Snapshot=t2.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=/irj/services/laf/themes/portal/sap_mango_polarwind/.., ENDITEM,
        LAST);
```

Creating a SAP-Web Vuser Script

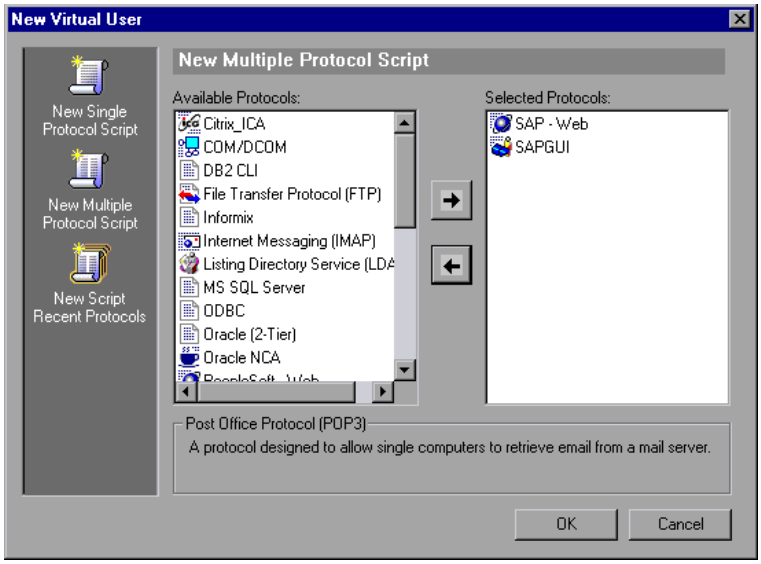
The first step in creating a SAP-Web Vuser script, is choosing the Vuser and script type. The **SAP-Web** Vuser is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script. In addition, you can use the single-protocol SAP (Click and Script) Vuser type. The SAP (Click and Script) Vuser generates a higher level, more intuitive script.

To create a SAP-Web Vuser:

1. Invoke VuGen and select **File** > **New**.
2. To record a session that does not incorporate any SAPGUI controls within the Workplace or Portal clients, create a single-protocol Vuser script using the **SAP-Web** Vuser type.

3 To record a session that uses SAPGUI controls, create either:

- a single-protocol Vuser script, specifying the **SAP (Click and Script)** protocol.
- a multi-protocol Vuser script, specifying both the **SAP-Web** and **SAPGUI** Vuser types.



Understanding a SAP-Web Vuser Script

The SAP-Web Vuser script typically contains several SAP transactions which make up a business process. The business process consists of functions that emulate user actions. For information about these functions, see the Web functions in the *Online Function Reference* (**Help** > **Function Reference**).

The following example shows a typical recording for a SAP Portal client:

```
vuser_init()
{
    web_reg_find("Text=SAP Portals Enterprise Portal 5.0",
        LAST);

    web_set_user("junior{UserNumber}",
        lr_decrypt("3ed4cfe457afe04e"),
        "sonata.hplab.com:80");

    web_url("sapportal",
        "URL=http://sonata.hplab.com/sapportal",
        "Resource=0",
        "RecContentType=text/html",
        "Snapshot=t1.inf",
        "Mode=HTML",
        EXTRARES,

"Url=/SAPPortal/IE/Media/sap_mango_polarwind/images/header/branding_image.jpg",
"Referer=http://sonata.hplab.com/hrnp$30001/sonata.hplab.coml:80/Action/26011[hea
der]", ENDITEM,
        "Url=/SAPPortal/IE/Media/sap_mango_polarwind/images/header/logo.gif",
"Referer=http://sonata.hplab.com/hrnp$30001/sonata.hplab.com:80/Action/26011[head
er]", ENDITEM,
…
        LAST);
```

The following section illustrates a multi-protocol recording in which the Portal client opens a SAP control. Note the switch from web_xxx to sapgui_xxx functions.

```
web_url("dummy",

"URL=http://sonata.hplab.com:1000/hrnp$30000/sonata.hplab.com:1000/Action/dumm
y?PASS_PARAMS=YES&dummyComp=dummy&Tcode=VA01&draggable=0&CompF
Name=VA01&Style=sap_mango_polarwind",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.hplab.com/sapportal",
        "Snapshot=t9.inf",
        "Mode=HTML",
        LAST);

    sapgui_open_connection_ex(" /H/Protector/S/3200 /WP",
        "",
        "con[0]");

    sapgui_select_active_connection("con[0]");

    sapgui_select_active_session("ses[0]");

    /*Before running script, enter password in place of asterisks in logon function*/

    sapgui_logon("JUNIOR{UserNumber}",
        "ides",
        "800",
        "EN",
        BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui102",
        END_OPTIONAL);
```

Replaying a SAP-Web Vuser Script

After creating and enhancing your SAP-Web Vuser script, you configure its run-time settings and run it from VuGen to check its functionality.

Run-Time settings let you control the Vuser behavior during replay. You configure these settings before running the Vuser script. You can set both General and Web related run-time settings.

The General settings include the run logic, pacing, logging, think time, and performance preferences. For information about the General run-time settings, see Chapter 79, "Configuring Run-Time Settings." For SAP-Web specific settings, see Chapter 80, "Configuring Network Run-Time Settings".

Once you configure the Run-Time settings, you save the Vuser script and run it from VuGen as a standalone test, to verify that it runs correctly. For further information, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After verifying that your Vuser script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

61

Siebel-Web Protocol

You use VuGen to record the activity in a Siebel Web environment and generate a Vuser script. When you run the script, Vusers emulate the actions within your Siebel environment.

This chapter includes:

About Developing Siebel-Web Vuser Scripts

The Siebel-Web protocol is similar to the standard Web Vuser, with several changes in the default settings to allow it to work with the Siebel Customer Relationship Management (CRM) application.

You record typical activities in your Siebel session. VuGen records the actions and generates functions with a **web_** prefix, that emulate the actions.

The sections below provide tips for working with Siebel-Web recorded Vuser Scripts and provide samples of the parameters that need to be correlated.

Recording a Siebel-Web Session

When recording a Siebel-Web session, use the following guidelines:

To record a Siebel-Web Vuser script:

1. Create a Siebel-Web type Vuser script from the ERP category.
2. Set the following Recording Options:
   - Record node: **HTML based script**

     Advanced HTML - Script options: **a script containing explicit URLs only**

     Advanced HTML - Non HTML-generated elements: **Do not record**
   - Advanced node: Clear the **Reset context for each action** option.
3. Record the login in the **vuser_init** section.
4. Record the Business Process in **Action1**.
5. Record the logout in the **vuser_end** section.
6. In the Run-Time settings, clear the **Simulate a new user on each iteration** option in the Browser Emulation node.

For more information on how to configure the Recording Options and Web related Run-Time settings, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168, and Chapter 80, "Configuring Network Run-Time Settings".

Correlating Siebel-Web Scripts

When creating a test script for a Siebel session, you will most probably need to use correlation in your script. Correlation is the mechanism by which VuGen saves dynamic values to parameters during record and replay, for use at a later point in the script. If you replayed the recorded script as is, without correlation, it would fail, since the values of the arguments differ each time the script runs. An example of such variables are **SWECount** and **SWEBMC**.

When you use correlation, VuGen saves the dynamic variables during both record and replay, and uses them at the appropriate points within the script. You can instruct VuGen to apply correlation during recording using one of the following methods:

- Siebel Correlation Library

  The Siebel correlation library automatically correlates most of the dynamic values, creating a concise script that you can replay without major modifications. This is the recommended method for correlation.

- VuGen Native Siebel Correlation

  The native, built-in rules, work on a low level, allowing you to debug your script and understand the correlations in depth.

Siebel Correlation Library

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

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

To enable correlation with this library:

1. Copy the DLL file into the bin directory of the product installation.
2. Open a multi-protocol script using the **Siebel-Web** Vuser type.

3 Enable UTF-8 support in the recording options. For more information, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168.

4 Open the recording option's Correlation node and click **Import**. Import the rules file, **WebSiebel77Correlation.cor**, from the \dat\webrulesdefaultsetting directory. If you are prompted with warnings, click **Override**. For more information, see "Setting the Correlation Recording Options" on page 1166.

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

When using the Siebel correlation library, verify that the SWE count rules (where the left boundary contains the **SWEC** string) are not disabled. For more information, see "Disabling and Enabling Rules" on page 968.

VuGen Native Siebel Correlation

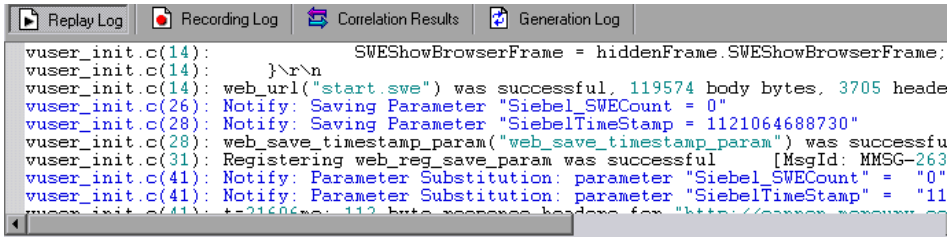
VuGen's native built-in rules for the Siebel server detect the Siebel server variables and strings, automatically saving them for use at a later point within the script.

You can view these rules in the list of correlation rules (see "Using VuGen's Correlation Rules" on page 854). The rules list the boundary criteria that are unique for Siebel server strings.

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

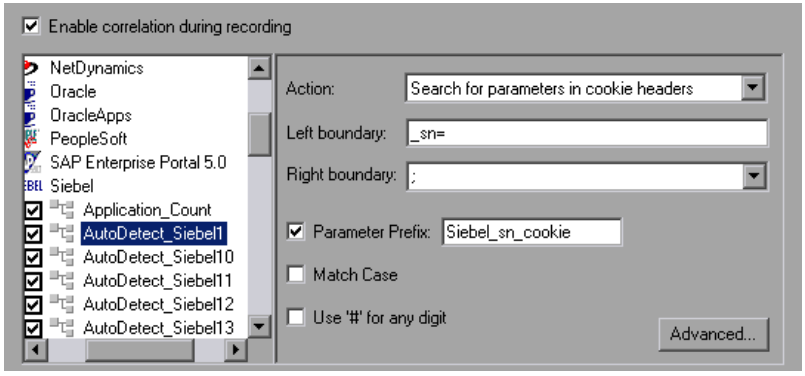
You can also create your own rules by entering unique boundary criteria in the Correlation Recording Options (see Chapter 53, "Web (HTTP/HTML) Correlation Rules") or after the recording from the Correlation Results tab (see "Performing a Scan for Correlations" on page 871).

In the Replay Log tab, VuGen indicates when it registers, saves, or uses the parameters. Note that to display this information, you need to enable Extended logging. For more information, see "Configuring the Log Run-Time Settings" on page 1260.



Simple Variable Correlation

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



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

```
/* Registering parameter(s) from source
web_reg_save_param("Siebel_sn_cookie6",
"LB/IC=_sn=",
"RB/IC=;",
"Ord=1",
"Search=headers",
"RelFrameId=1",
LAST);

…

web_url("start.swe_3",
"URL=http://cannon.hplab.com/callcenter_enu/start.swe?SWECmd=GotoPostedAction
&SWEDIC=true&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECount}&SWEFrame
=top._sweclient&SWECS=true",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://cannon.hplab.com/callcenter_enu/start.swe?SWECmd=GetCachedFra
me&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECount}&SWEFrame=top._swe",
"Snapshot=t4.inf",
"Mode=HTML",
LAST);
```

Function Correlation

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

<parameter_name> = <recorded_value> (display_name)

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

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

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

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

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

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

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

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

Note: Correlation for the **SWEC** parameter is not done through the correlation rules. VuGen handles it automatically with a built-in detection mechanism. For more information, see "SWEC Correlation" on page 969.

Disabling and Enabling Rules

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

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

To disable rules:

1. Open the Correlation recording options. Select **Tools** > **Recording Options** and click the Correlation node.
2. Select the **Enable correlation during recording** option. The dialog box displays the supported servers.
3. Expand the rules under Siebel and view their properties.
4. Clear the check box adjacent to the rule for each rule you want to disable.

SWEC Correlation

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

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

or

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

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

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

```
Siebel_SWECount_var += 1;

lr_save_int(Siebel_SWECount_var, "Siebel_SWECount");

web_submit_data("start.swe_8",
    "Action=http://cannon.hplab.com/callcenter_enu/start.swe",
    "Method=POST",
    "TargetFrame=",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t9.inf",
    "Mode=HTML",
    "EncodeAtSign=YES",
    ITEMDATA,
    "Name=SWERPC", "Value=1", ENDITEM,
    "Name=SWEC", "Value={Siebel_SWECount}", ENDITEM,
    "Name=SWECmd", "Value=InvokeMethod", ENDITEM,
    "Name=SWEService", "Value=SWE Command Manager", ENDITEM,
    "Name=SWEMethod", "Value=BatchCanInvoke", ENDITEM,
    "Name=SWEIPS",…
    LAST);
```

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

Correlating SWECount, ROWID, and SWET Parameters

This section provides tips for correlating several special parameters:

- SWECount
- Row ID Length
- SWETS (Timestamps)

SWECount

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

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

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

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

Row ID Length

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

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

If you define parameters to correlate the string as

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

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

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

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

SWETS (Timestamps)

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

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

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

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

Troubleshooting Siebel-Web Vuser Scripts

This section provides information about errors you might encounter when creating a script, and the breakdown diagnostic tool.

- Typical Errors
- Recording Breakdown Information

Typical Errors

You may encounter one or more of the following errors while creating a Siebel-Web Vuser script:

- Back or Refresh Error
- Same Values
- No Content HTTP Response
- Restoring the Context
- Cannot Locate Record
- End of File
- Unable to Retrieve Search Categories

Back or Refresh Error

An error message relating to **Back or Refresh** typically has the following text:

We are unable to process your request. This is most likely because you used the browser BACK or REFRESH button to get to this point.

Cause: The possible causes of this problem may be:

- The SWEC was not correlated correctly for the current request.
- The SWETS was not correlated correctly for the current request.
- The request was submitted twice to the Siebel server without the SWEC being updated.

- A previous request should have opened a frame for the browser to download. This frame was not created on the server probably because the SWEMethod has changed since the recording.

Same Values

A typical Web page response to the **Same Values** error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`10`0`1`Errors`0`2`0`Level0`0`ErrMsg`Th
e same values for 'Name' already exist. If you would like to enter a new record,
please make sure that the field values are unique.`ErrCode`28591`
```

Cause: The possible cause of this problem may be that one of the values in the request (in the above example, the value of the Name field) duplicates a value in another row of the database table. This value needs to be replaced with a unique value to be used for each iteration per user. The recommended solution is to replace the row ID with its parameter instead insuring that it will be unique.

No Content HTTP Response

A typical HTTP response for a **No Content HTTP Response** type error is:

```
HTTP/1.1 204 No Content
Server: Microsoft-IIS/5.0
Date: Fri, 31 Jan 2003 21:52:30 GMT
Content-Language: en
Cache-Control: no-cache
```

Cause: The possible causes of this problem may be that the row ID is not correlated at all or that it is correlated incorrectly.

Restoring the Context

The typical Web page response to the **Restoring the Context** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`9`0`1`Errors`0`2`0`Level0`0`ErrMsg`An
error happened during restoring the context for requested
location`ErrCode`27631`
```

Cause: The possible causes of this problem may be that the rowid is not correlated or that it is correlated incorrectly.

Cannot Locate Record

The typical Web page response to the **Cannot locate record** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`23`0`2`Errors`0`2`0`Level0`0`ErrMsg`Ca
nnot locate record within view: Contact Detail - Opportunities View applet:
Opportunity List Applet.`ErrCode`27573`
```

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

End of File

The typical Web page response to the **End of File** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`28`0`1`Errors`0`2`0`Level0`0`ErrMsg`An
end of file error has occurred. Please continue or ask your systems administrator
to check your application configuration if the problem persists.`ErrCode`28601`
```

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

Unable to Retrieve Search Categories

The typical Web page response to the **Unable to Retrieve Search Categories** type error is:

Cause: A possible cause of this problem may be that the search frame was not downloaded from the server. This occurs when the previous request did not ask the server to create the search frame correctly.

Recording Breakdown Information

VuGen provides a diagnostic tool for understanding the transaction components in your test—**transaction breakdown**. Using transaction breakdown, you can determine where the bottlenecks are and the issues that need to be resolved.

When preparing your script for transaction breakdown, we recommend that you add think time at the end of each transaction using the ratio of one second per hour of testing. For more information on entering think time, see Chapter 6, "Enhancing Vuser Scripts."

In order to record the transaction breakdown information, you need to modify your the parameterization functions in your script.

To prepare your script for transaction breakdown:

1 Identify the script parameterization replacement of the Session ID.

```
/* Registering parameter(s) from source task id 15
    // {Siebel_sn_body4} = "28eMu9uzkn.YGFFevN1FdrCfCCOc8c_"
    // */
    web_reg_save_param("Siebel_sn_body4",
        "LB/IC=_sn=",
        "RB/IC=&",
        "Ord=1",
        "Search=Body",
        "RelFrameId=1",
        LAST);
```

2 Mark the next **web_submit_data** function as a transaction by enclosing it with **lr_start_transaction** and **lr_end_transaction** functions.

3 Before the end of the transactions, add a call to **lr_transaction_instance_add_info**, where the first parameter, 0 is mandatory and the session ID has a SSQLBD prefix.

```
lr_start_transaction("LoginSQLSync");
    web_submit_data("start.swe_2",
        "Action=http://design/callcenter_enu/start.swe",
        "Method=POST",
        "RecContentType=text/html",
        "Referer=http://design/callcenter_enu/start.swe",
        "Snapshot=t2.inf",
        "Mode=HTML",
        ITEMDATA,
        "Name=SWEUserName", "Value=wrun", ENDITEM,
        "Name=SWEPassword", "Value=wrun", ENDITEM,
        "Name=SWERememberUser", "Value=Yes", ENDITEM,
        "Name=SWENeedContext", "Value=false", ENDITEM,
        "Name=SWEFo", "Value=SWEEntryForm", ENDITEM,
        "Name=SWETS", "Value={SiebelTimeStamp}", ENDITEM,
        "Name=SWECmd", "Value=ExecuteLogin", ENDITEM,
        "Name=SWEBID", "Value=-1", ENDITEM,
        "Name=SWEC", "Value=0", ENDITEM,
        LAST);

lr_transaction_instance_add_info(0,lr_eval_string("SSQLBD:{Siebel_sn_body4}"));
lr_end_transaction("LoginSQLSync", LR_AUTO);
```

Note: To avoid session ID conflicts, make sure that the Vusers log off from the database at the end of each session.

62

RTE Protocol

RTE Vusers operate terminal emulators in Windows environments. This chapter describes how to develop Windows-based RTE Vuser scripts.

This chapter includes:

About Developing RTE Vuser Scripts

RTE Vusers operate terminal emulators in order to load test client/server systems.

You record a terminal emulator session with VuGen to represent a true user's actions. You can then enhance your recorded script with transaction and synchronization functions.

This chapter describes how to develop Windows-based RTE Vuser scripts.

Introducing RTE Vusers

An RTE Vuser types character input into a terminal emulator, submits the data to a server, and then waits for the server to respond. For instance, suppose that you have a server that maintains customer information for a maintenance company. Every time a field service representative makes a repair, he accesses the server database by modem using a terminal emulator. The service representative accesses information about the customer and then records the details of the repair that he performs.

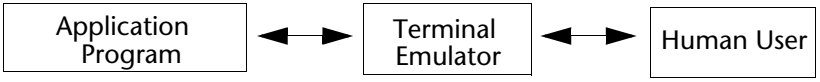
You could use RTE Vusers to emulate this case. An RTE Vuser would:

1. Type **60** at the command line to open an application program.
2. Type **F296**, the field service representative's number.
3. Type **NY270**, the customer number.
4. Wait for the word "Details" to appear on the screen. The appearance of "Details" indicates that all the customer details are displayed on the screen.
5. Type **Changed gasket P249, and performed Major Service** the details of the current repair.
6. Type **Q** to close the application program.

You use VuGen to create RTE Vuser scripts. The script generator records the actions of a human user in a terminal emulator. It records the keyboard input from the terminal window, generates the appropriate statements, and inserts them into the Vuser script. While you record, the script generator automatically inserts synchronization functions into the script. For details, see Chapter 64, "RTE - Synchronization."

Understanding RTE Vuser Technology

An RTE Vuser emulates the actions of a real user. Human users use terminals or terminal emulators to operate application programs.



In the RTE Vuser environment, a Vuser replaces the human. The Vuser operates PowerTerm, a terminal emulator.

PowerTerm works like a standard terminal emulator, supporting common protocols such as IBM 3270 & 5250, VT100, VT220, and VT420-7.

Getting Started with RTE Vuser Scripts

This section provides an overview of the process of developing RTE Vuser scripts using VuGen.

To develop an RTE Vuser script:

1 Record the basic script using VuGen.

Use the Virtual User Generator (VuGen) to record the operations that you perform in a terminal emulator. VuGen records the keyboard input from the terminal window, generates the appropriate statements, and then inserts these statements into the Vuser script.

For details, see Chapter 63, "RTE - Recording."

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, synchronization functions, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

4 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

5 Run the script from VuGen.

Run the script from VuGen to verify that it runs correctly. View the standard output to verify that the program is communicating properly with the server.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using TE Functions

The functions developed to emulate a terminal communicating with a server are called TE Vuser functions. Each TE Vuser function has a **TE** prefix. VuGen automatically records most of the TE functions listed in this section during an RTE session. You can also manually program any of the functions into your script.

The TE functions are divided into the following categories: Terminal Emulator Connection, Text Retrieval, Cursor, System Variable, Error Code, Typing, and Synchronization Functions.

You can also manually program any of the functions into your Vuser script. In text view, you can manually add new functions utilizing the Intellisense and Complete Function features. In Tree view, select **Insert** > **New Step** and select the desired step.

For syntax and examples of the TE functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Working with Ericom Terminal Emulation

VuGen supports record and replay with Ericom Terminal Emulators.

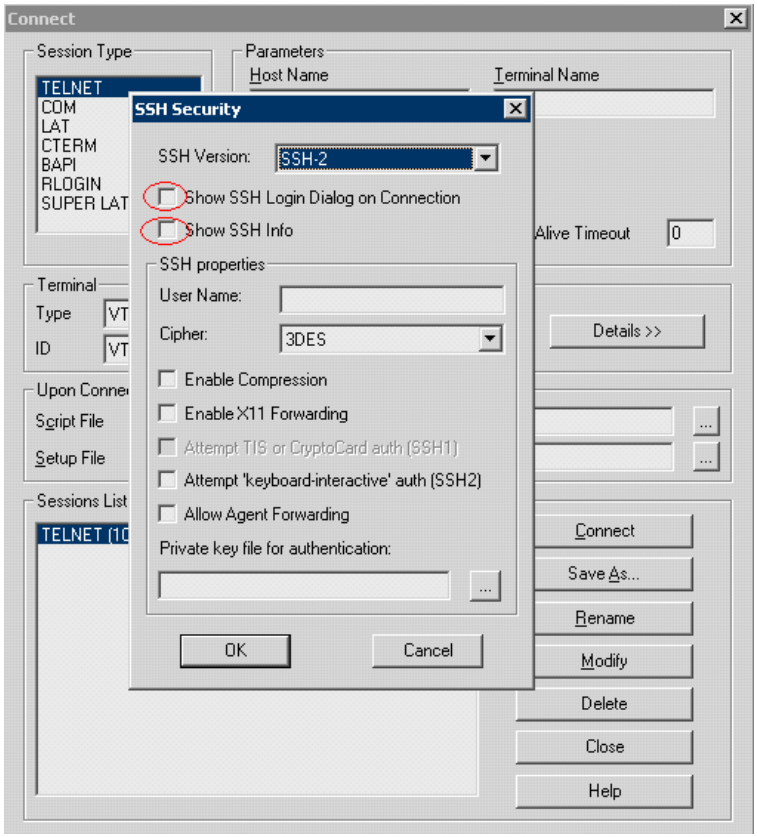
The Ericom support handles escape sequences during record and replay. Ericom's PowerTerm lets you map PC keys to custom escape sequences. For information about mapping, see the PowerTerm help.

When a user presses mapped keys while recording an Ericom VT session, VuGen generates **TE_send_text** functions instead of the standard **TE_type**. This allows the script to handle custom escape sequences in a single step. For more information, see the *Online Function Reference* (**Help** > **Function Reference**) for the **TE_send_text** function.

SSL and SSH Support for Ericom

VuGen also supports SSL/SSH record and replay for the RTE Ericom library. To work with SSL or SSH, you select the type in the **Security** section of the Connect dialog box.

When working with SSH Security, by default VuGen opens a popup dialog box prompting you for more information. We recommend that you disable the **Show** options to prevent the popups from being issued. If you enable these popups, it may affect the replay. You can access the advanced security options by clicking the **Details** button.

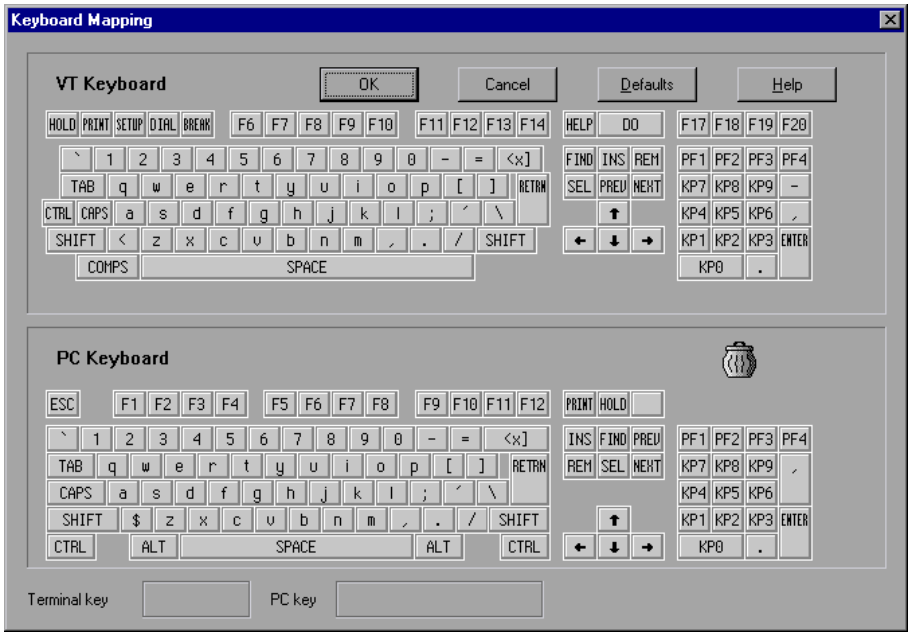


Mapping Terminal Keys to PC Keyboard Keys

Because you are using a terminal emulator, you will be using a PC keyboard in place of a terminal keyboard. Many keys that are found on the terminal keyboard are not available on a PC keyboard. Examples of such keys are HELP, AUTOR, and PUSH, which are found on the IBM 5250 keyboard. To successfully operate the terminal emulator and any associated application programs, you may have to map certain terminal keys to keys on the PC keyboard.

To map a terminal key to a key on the PC keyboard:

1 In the terminal emulator, select **Options** > **Keyboard Map**, or click the **Keyboard Mapping** button. The Keyboard Mapping dialog box opens.



2 Click the Keyboard **Mapping** button on the toolbar. To map a terminal key to a PC key, drag a key from the upper terminal keyboard to a PC key on the lower keyboard.

You can click the Shift and/or Control keys on the upper keyboard to display additional key functions that can be viewed only by first selecting either of these keys. You can then drag the required key from the upper terminal keyboard to a key on the lower PC keyboard.

To cancel a definition, drag the PC key definition to the wastebasket. This restores the default function of the PC key.

To restore the default mappings, click **Defaults**.

63

RTE - Recording

You use VuGen to record Windows-based Remote Terminal Emulation (RTE) Vuser scripts.

This chapter includes:

About Recording RTE Vuser Scripts

You use VuGen to record Windows-based RTE Vuser scripts. VuGen uses the PowerTerm terminal emulator to emulate a wide variety of terminal types. You use PowerTerm to perform a typical terminal connection, followed by typical business processes. Thereafter, you perform the log off procedure. While you perform typical user actions in the terminal emulator, VuGen generates the appropriate statements, and inserts them into a Vuser script. You can view and edit the script while recording.

Before recording an RTE Vuser script, make sure that the recording options are set correctly. The recording options allow you to control how VuGen generates certain functions while you record a Vuser script. VuGen applies the recording options during all subsequent recording sessions.

Creating a New RTE Vuser Script

Before recording a user's actions into a Vuser script, you must open one. You can open an existing script, or create a new one. You use VuGen to create a new Vuser script.

To create a new RTE Vuser script:

1 Select **Virtual User Generator** from your product's start menu. The VuGen window opens.

2 Click the **New** button. The **New Virtual User** dialog box opens:

3 Select the **Legacy** protocol type, and select **Terminal Emulator (RTE).** Click **OK**. VuGen generates and displays an empty RTE script, with the cursor positioned to begin recording in the **vuser_init** section.

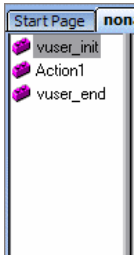
Recording the Terminal Setup and Connection Procedure

After you create a skeleton Vuser script, you record the terminal setup and connection procedure into the script. VuGen uses the PowerTerm terminal emulator when you record an RTE Vuser script.

To record the terminal setup and connection procedure:

1 Open an existing RTE Vuser script, or create a new one.

2 In the **Sections** box, select the section into which you want VuGen to insert the recorded statements. The available sections are **vuser_init**, **Actions**, and **vuser_end**.

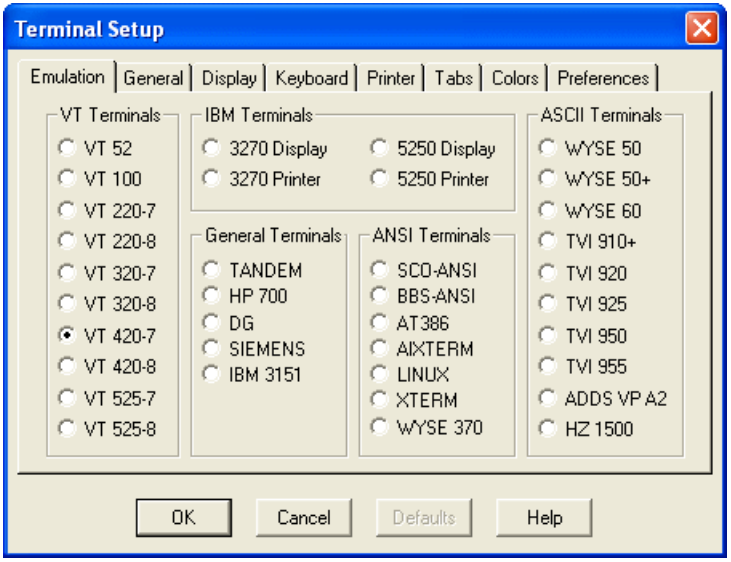


Note: Always record the terminal setup and connection procedure into the **vuser_init** section. The **vuser_init** section is not repeated when you run multiple iterations of a Vuser script—only the **Actions** section is repeated. For more information on the iteration settings, see Chapter 79, "Configuring Run-Time Settings."

3 In the Vuser script, place the cursor at the location where you want to begin recording.

4 Click the **Record** button. The PowerTerm main window opens.

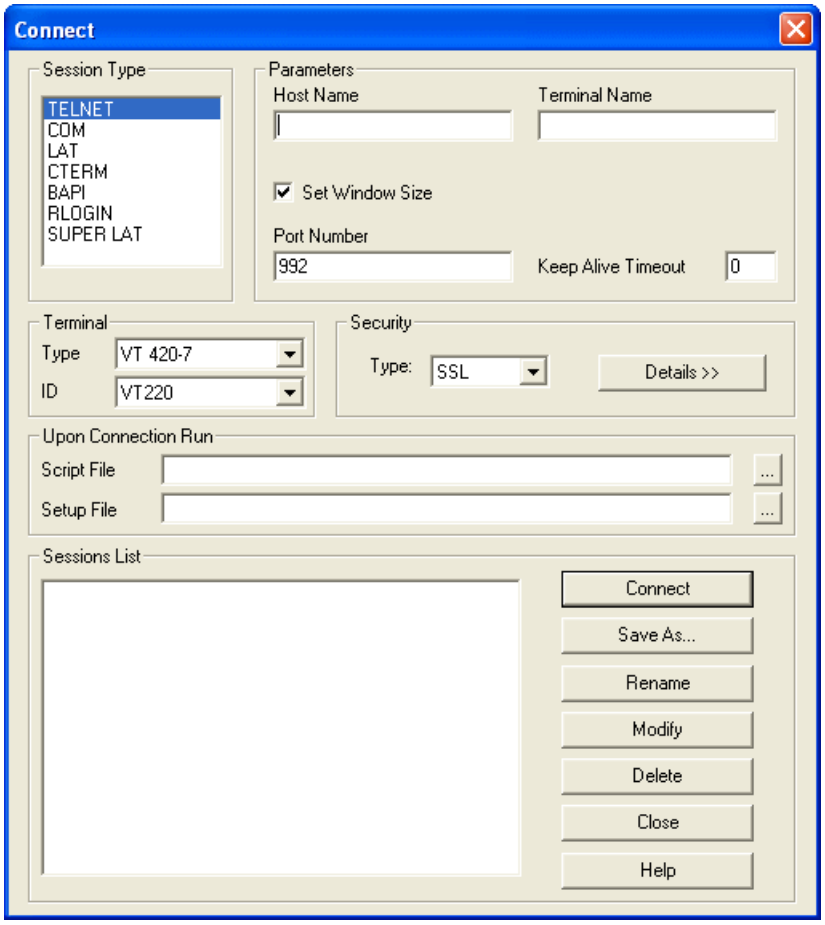
5 From the PowerTerm menu bar, select **Terminal** > **Setup** to display the Terminal Setup dialog box.



6 Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.

Note: Select an IBM terminal type to connect to an AS/400 machine or an IBM mainframe; select a VT terminal type to connect to a UNIX workstation.

7 Select **Communication** > **Connect** to display the Connect dialog box.



8 Under **Session Type**, select the type of communication to use.

9 Under **Parameters**, specify the required options. The available parameters vary depending on the type of session that you select. For details on the parameters, click **Help**.

Tip: Click **Save As** to save the parameter-sets for re-use in the future. The parameter-sets that you save are displayed in the Sessions List box.

10 Click **Connect**. PowerTerm connects to the specified system, and VuGen inserts a **TE_connect** function into the script, at the insertion point. The **TE_connect** statement has the following form:

```
/* *** The terminal type is VT 100. */
TE_connect(
    "comm-type = telnet;"
    "host-name = alfa;"
    "telnet-port = 992;"
    "terminal-id = ;"
    "set-window-size = true;"
    "security-type = ssl;"
    "ssl-type = tls1;"
    "terminal-type = vt100;"
    "terminal-model = vt100;"
    "login-command-file = ;"
    "terminal-setup-file = ;"
    , 60000);
if (TE_errno != TE_SUCCESS)
    return -1;
```

The inserted **TE_connect** statement is followed by an if statement that checks whether or not the **TE_connect** function succeeds during replay.

Note: Do not record more than one connection to a server **(TE_connect)** in a Vuser script.

The terminal setup and connection procedure is complete. You are now ready to begin recording typical user actions into the Vuser script, as described below.

Recording Typical User Actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the **Actions** section of the Vuser script. Only the **Actions** section of a Vuser script is repeated when you run multiple iterations of the script. For details on setting iterations, see Chapter 79, "Configuring Run-Time Settings."

When recording a session, VuGen records the text strokes and not the text. Therefore, it is not recommended that you copy and paste commands into the PowerTerm window—instead, type them in directly.

To record user actions:

1. Open an existing RTE Vuser script, and then click **Actions** in the **Section** box.
2. Proceed to perform typical user actions in the terminal emulator. VuGen generates the appropriate statements, and inserts them into the Vuser script while you type. If necessary, you can edit the recorded statements while you record the script.

   Note: By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function. To change the waiting time, see "RTE Recording Options" on page 1158.

When you finish recording the typical user actions, proceed to record the log off procedure, as described in the next section.

Recording the Log Off Procedure

You record the Vuser log off process into the **vuser_end** section of the Vuser script. The **vuser_end** section is not repeated when you run many iterations of the script. For details on setting iterations, see Chapter 79, "Configuring Run-Time Settings."

To record the log off procedure:

1. Make sure that you have performed and recorded the typical user actions as described in the previous section.
2. In the VuGen main window, click **vuser_end** in the **Section** box.

3 Perform the log off procedure. VuGen records the procedure into the **vuser_end** section of the script.

4 Click **Stop Recording on** the Recording toolbar. The main VuGen window displays all the recorded statements.

5 Click **Save to** save the recorded session. The Save As dialog box opens (for new Vuser scripts only). Specify a script name. After recording a script, you can manually edit it in VuGen's main window.

Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to "type" character input into the PowerTerm terminal emulator:

- **TE_type** sends characters to the terminal emulator. When recording, the VuGen automatically generates **TE_type** functions for keyboard input to the terminal window. For details, see "Using the TE_type Function" on page 992.
- **TE_typing_style** determines the speed at which the Vuser types. You can manually define the typing style by inserting a **TE_typing_style** function into the Vuser script. For details, see "Setting the Typing Style" on page 994. Alternatively, you can set the typing style by using the run-time settings. See Chapter 81, "Run-Time Settings for Selected Protocols" for details.

Note: While recording an RTE Vuser script, do not use the mouse to relocate the cursor within the terminal emulator window. VuGen does not record these cursor movements.

Using the TE_type Function

When you record a script, the VuGen records all keyboard input and generates appropriate **TE_type** functions. During execution, **TE_type** functions send formatted strings to the terminal emulator.

Keyboard input is defined as a regular text string (including blank spaces). For example:

```
TE_type ("hello, world");
```

Input key names longer than one character are represented by identifiers beginning with the letter *k*, and are bracketed within greater-than/less-than signs (< >).

For example, the following function depicts the input of the Return key followed by the Control and y keys:

```
TE_type("<kReturn><kControl-y>");
```

Some other examples include: <kF1>, <kUp>, <kF10>, <kHelp>, <kTab>.

To determine a key name, record an operation on the key, and then check the recorded statement for its name.

Note: When you program a **TE_type** statement (rather than recording it), use the key definitions provided in the *Online Function Reference* (**Help > Function Reference**).

Setting the Timeout Value for TE_type

If a Vuser attempts to submit a **TE_type** statement while the system is in X SYSTEM (or input inhibited) mode, the Vuser will wait until the X SYSTEM mode ends before typing. If the system stays in X SYSTEM mode for more than TE_XSYSTEM_TIMEOUT milliseconds, then the **TE_type** function returns a TE_TIMEOUT error.

You can set the value of TE_XSYSTEM_TIMEOUT by using **TE_setvar**. The default value for TE_XSYSTEM_TIMEOUT is 30 seconds.

Allowing a Vuser to Type Ahead

Under certain circumstances you may want a Vuser to submit a keystroke even though the system is in X SYSTEM (or input inhibited) mode. For example, you may want the Vuser to press the Break key. You use the TE_ALLOW_TYPEAHEAD variable to enable the Vuser to submit a keystroke even though the system is in X SYSTEM mode.

Set TE_ALLOW_TYPEAHEAD to zero to disable typing ahead, and to any non-zero number to permit typing ahead. You use **TE_setvar** to set the value of TE_ALLOW_TYPEAHEAD. By default, TE_ALLOW_TYPEAHEAD is set to zero, preventing keystrokes from being sent during X SYSTEM mode.

For more information about the **TE_type** function and its conventions, see the *Online Function Reference* (**Help > Function Reference**).

Setting the Typing Style

You can set two typing styles for RTE Vuser: FAST and HUMAN. In the FAST style, the Vuser types input into the terminal emulator as quickly as possible. In the HUMAN style, the Vuser pauses after typing each character. In this way, the Vuser more closely emulates a human user typing at the keyboard.

You set the typing style using the **TE_typing_style** function. The syntax of the **TE_typing_style** function is:

int **TE_typing_style (**char ******style***);**

where *style* can be FAST or HUMAN. The default typing style is HUMAN. If you select the HUMAN typing style, the format is:

HUMAN, *delay* [,*first_delay*]

The delay indicates the interval (in milliseconds) between keystrokes. The optional parameter first_delay indicates the wait (in milliseconds) before typing the first character in the string. For example,

```
TE_typing_style ("HUMAN, 100, 500");
TE_type ("ABC");
```

means that the Vuser will wait 0.5 seconds before typing the letter A; it will then wait 0.1 seconds before typing "B" and then a further 0.1 seconds before typing "C".

For more information about the **TE_typing_style** function and its conventions, see the *Online Function Reference* (**Help** > **Function Reference**).

In addition to setting the typing style by using the **TE_typing_style** function, you can also use the run-time settings. For details, see Chapter 81, "RTE Run-Time Settings."

Generating Unique Device Names

Some protocols, such as APPC, require a unique device name for each terminal that logs on to the system. Using the run-time settings, you can specify that the **TE_connect** function generate a unique 8-character device name for each Vuser, and connect using this name. Although this solves the requirement for uniqueness, some systems have an additional requirement: The device names must conform to a specific format. See Chapter 79, "Configuring Run-Time Settings" for more information.

To define the format of the device names that the **TE_connect** function uses to connect a Vuser to the system, add an **RteGenerateDeviceName** function to the Vuser script. The function has the following prototype:

```
void RteGenerateDeviceName(char buf[32])
```

The device name should be written into **buf**.

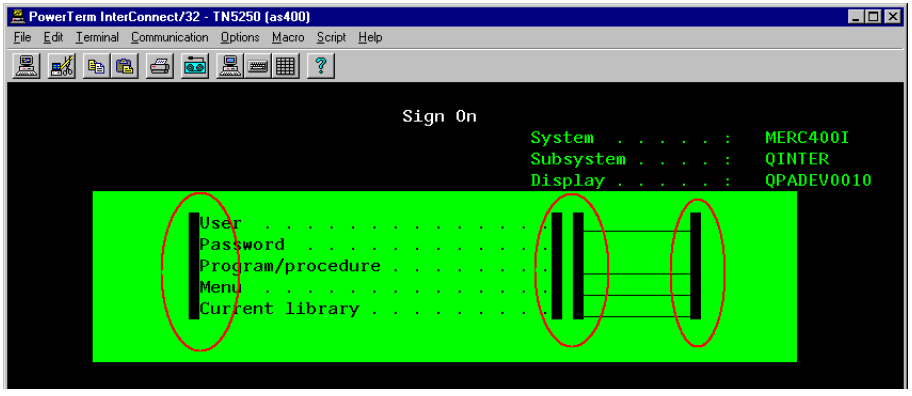
If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

In the following example, the RteGenerateDeviceName function generates unique device names with the format "TERMx". The first name is TERM0, followed by TERM1, TERM2, and so forth.

```
RteGenerateDeviceName(char buf[32])
{
    static int n=0;
    sprintf(buf, "TERM%d", n);
    n=n+1;
}
```

Setting the Field Demarcation Characters

Some terminal emulators use demarcation characters to mark the beginning and the end of each field. These demarcation characters are not visible—appearing on the screen as spaces. In the terminal emulator shown below, the colors in the middle section of the screen have been inverted to display the field demarcation characters. These characters are surrounded by ellipses.



The **TE_wait_text**, **TE_get_text**, and **TE_find_text** functions operate by identifying the characters in a specified portion of the screen. If a field demarcation character is located within the specified section, you can identify the character as a space or an ASCII character. You use the TE_FIELD_CHARS system variable to specify the method of identification. You can set TE_FIELD_CHARS to 0 or 1:

- 0 specifies that the character in the position of the field demarcation characters is returned as a space.
- 1 specifies that the character in the position of the field demarcation characters is returned as an ascii code (ascii 0 or ascii 1).

By default, TE_FIELD_CHARS is set to 0.

You retrieve and set the value of TE_FIELD_CHARS by using the **TE_getvar** and **TE_setvar** functions.

64

RTE - Synchronization

Synchronization functions in an RTE Vuser script help you synchronize the input that a Vuser submits to a terminal emulator with the responses from the server.

This chapter includes:

About Synchronizing Vuser Scripts

Depending on the system you are testing, you may need to synchronize the input that a Vuser sends to a terminal emulator with the subsequent responses from the server. When you synchronize input, you instruct the Vuser to suspend script execution and wait for a cue from the system, before the Vuser performs its next action. For instance, suppose that a human user wants to submit the following sequence of key strokes to a bank application:

1. Type 1 to select "Financial Information" from the menu of a bank application.
2. When the message "What information do you require?" appears, type 3 to select "Dow Jones Industrial Average" from the menu.
3. When the full report has been written to the screen, type 5 to exit the bank application.

In this example, the input to the bank application is synchronized because at each step, the human user waits for a visual cue before typing. This cue can be either the appearance of a particular message on the screen, or stability of all the information on the screen.

You can synchronize the input of a Vuser in the same way by using the TE-synchronization functions, **TE_wait_sync**, **TE_wait_text**, **TE_wait_silent**, and **TE_wait_cursor**. These functions effectively emulate a human user who types into a terminal window and then waits for the server to respond, before typing in the next command.

The **TE_wait_sync** function is used to synchronize block-mode (IBM) terminals only. The other TE-synchronization functions are used to synchronize character-mode (VT) terminals.

When you record an RTE Vuser script, VuGen can automatically generate and insert **TE_wait_sync**, **TE_wait_text**, and **TE_wait_cursor** statements into the script. You use VuGen's recording options to specify which synchronization functions VuGen should insert.

Note: Do not include any synchronization statements in the *Vuser_end* section of a Vuser script. Since a Vuser can be aborted at any time, you cannot predict when the *Vuser_end* section will be executed.

Synchronizing Block-Mode (IBM) Terminals

The **TE_wait_sync** function is used for synchronization RTE Vusers operating block-mode (IBM) terminals. Block-mode terminals display the "X SYSTEM" message to indicate that the system is in Input Inhibited mode. When a system is in the Input Inhibited mode no typing can take place because the terminal emulator is waiting for a transfer of data from the server.

When you record a script on a block-mode terminal, by default, VuGen generates and inserts a **TE_wait_sync** function into the script each time the "X SYSTEM" message appears. You use VuGen's recording options to specify whether or not VuGen should automatically insert **TE_wait_sync** functions.

When you run a Vuser script, the **TE_wait_sync** function checks if the system is in the X SYSTEM mode. If the system is in the X SYSTEM mode, the **TE_wait_sync** function suspends script execution. When the "X SYSTEM" message is removed from the screen, script execution continues.

Note: You can use the **TE_wait_sync** function only with IBM block-mode terminals emulators (5250 and 3270).

In general, the **TE_wait_sync** function provides adequate synchronization for all block-mode terminal emulators. However, if the **TE_wait_sync** function is ineffective in a particular situation, you can enhance the synchronization by including a **TE_wait_text** function. For more information on the **TE_wait_text** function, see "Waiting for Text to Appear on the Screen" on page 1006, and the *Online Function Reference* (**Help > Function Reference**).

The syntax of the **TE_wait_sync** function is:

```
TE_wait_sync ();
```

In the following script segment, the Vuser logs on with the user name "QUSER" and the password "HPLAB". The Vuser then presses Enter to submit the login details to the server. The terminal emulator displays the X SYSTEM message while the system waits for the server to respond.

The TE_wait_sync statement causes the Vuser to wait until the server has responded to the login request, that is, for the X SYSTEM message to be removed—before executing the next line of the script.

```
TE_type("QUSER");
lr_think_time(2);
TE_type("<kTab>HPLAB");
lr_think_time(3);
TE_type("<kEnter>");
TE_wait_sync();
....
```

When a **TE_wait_sync** function suspends the execution of a script while an X SYSTEM message is displayed, the Vuser continues to monitor the system—waiting for the X SYSTEM message to disappear. If the X SYSTEM message does not disappear before the synchronization timeout expires, the **TE_wait_sync** function returns an error code. The default timeout is 60 seconds.

To set the TE_wait_sync synchronization timeout:

1. Select **Vuser** > **Run-Time Settings**. The Run-Time Settings dialog box appears.
2. Select the **RTE:RTE** node in the Run-Time setting tree.
3. Under **X SYSTEM Synchronization**, enter a value (in seconds) in the **Timeout** box.
4. Click **OK** to close the Run-Time Settings dialog box.

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X SYSTEM mode. When the terminal returns from the X SYSTEM mode, the Vuser continues to monitor the system for a short period to verify that the terminal is fully stable, that is, that the system does not return to the X SYSTEM mode. Only then does the **TE_wait_sync** function terminate and allow the Vuser to continue executing its script. The period that the Vuser continues to monitor the system, after the system has returned from the X SYSTEM mode, is known as the stable time. The default stable time is 1000 milliseconds.

You may need to increase the stable time if your system exhibits the following behavior:

When a system returns from the X SYSTEM mode, some systems "flickers" to and from the X SYSTEM for a short period of time until the system stabilizes. If the system remains out of the X SYSTEM mode for more than one second, and then returns to the X SYSTEM mode, the **TE_wait_sync** function will assume that the system is stable. If a Vuser then tries to type information to the system, the system will shift into keyboard-locked mode.

Alternatively, if your system never flickers when it returns from the X SYSTEM mode, you can reduce the stable time to less than the default value of one second.

To change the stable time for TE_wait_sync functions:

1. Select **Vuser** > **Run-Time Settings**. The Run-Time Settings dialog box appears.
2. Select the **RTE:RTE** node.
3. Under **X SYSTEM Synchronization**, enter a value (in milliseconds) in the **Stable time** box.
4. Click **OK** to close the Run-Time Settings dialog box.

For more information on the **TE_wait_sync** function, see the *Online Function Reference* (**Help** > **Function Reference**).

You can instruct VuGen to record the time that the system remains in the X SYSTEM mode each time that the X SYSTEM mode is entered. To do so, VuGen inserts a TE_wait_sync_transaction function after each TE_wait_sync function, as shown in the following script segment:

```
TE_wait_sync();
TE_wait_sync_transaction("syncTrans1");
```

Each **TE_wait_sync_transaction** function creates a transaction with the name "default." This allows you to analyze how long the terminal emulator waits for responses from the server during a scenario run. You use the recording options to specify whether VuGen should generate and insert **TE_wait_sync_transaction** statements.

To instruct VuGen to insert TE_wait_sync_transaction statements:

1. Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.
2. Select the **Generate Automatic X SYSTEM transactions** option, and then click **OK**.

Synchronizing Character-Mode (VT) Terminals

There are three types of synchronization that you can use for character-mode (VT) terminals. The type of synchronization that you select depends on:

- the design of the application that is running in the terminal emulator
- the specific action to be synchronized

Waiting for the Cursor to Appear at a Specific Location

The preferred method of synchronization for VT type terminals is cursor synchronization. Cursor synchronization is particularly useful with full-screen or form-type applications, as opposed to scrolling or TTY-type applications.

Cursor synchronization uses the **TE_wait_cursor** function. When you run an RTE Vuser script, the **TE_wait_cursor** function instructs a Vuser to suspend script execution until the cursor appears at a specified location on the screen. The appearance of the cursor at the specified location means that the application is ready to accept the next input from the terminal emulator.

The syntax of the **TE_wait_cursor** function is:

int **TE_wait_cursor (**int *col*, int *row*, int *stable*, int *timeout*);

During script execution, the **TE_wait_cursor** function waits for the cursor to reach the location specified by *col*, *row*.

The **stable** parameter specifies the time (in milliseconds) that the cursor must remain at the specified location. If you record a script using VuGen, **stable** is set to 100 milliseconds by default. If the client application does not become stable in the time specified by the **timeout** parameter, the function returns TIMEOUT. If you record a script using VuGen, **timeout** is set by default to the value of TIMEOUT, which is 90 seconds. You can change the value of both the **stable** and **timeout** parameters by directly editing the recorded script.

The following statement waits for the cursor to remain stable for three seconds. If the cursor doesn't stabilize within 10 seconds, the function returns TIMEOUT.

```
TE_wait_cursor (10, 24, 3000, 10);
```

For more information on the **TE_wait_cursor** function, see the *Online Function Reference* (**Help > Function Reference**).

You can instruct VuGen to automatically generate TE_wait_cursor statements, and insert them into a script, while you record the script. The following is an example of a TE_wait_cursor statement that was automatically generated by VuGen:

```
TE_wait_cursor(7, 20, 100, 90);
```

To instruct VuGen to automatically generate TE_wait_cursor statements, and insert them into a script while recording:

1. Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.
2. Under **Generate Automatic Synchronization Commands** select the **Cursor** check box, and then click **OK**.

Waiting for Text to Appear on the Screen

You can use text synchronization to synchronize an RTE Vuser running on a VT terminal emulator. Text synchronization uses the **TE_wait_text** function. During script execution, the **TE_wait_text** function suspends script execution and waits for a specific string to appear in the terminal window before continuing with script execution. Text synchronization is useful with those applications in which the cursor does not consistently appear in a predefined area on the screen.

Note: Although text synchronization is designed to be used with character mode (VT) terminals, it can also be used with IBM block-mode terminals. Do not use automatic text synchronization with block-mode terminals.

The syntax of the **TE_wait_text** function is:

int **TE_wait_text (**char ***pattern*, int *timeout*, int *col1*, int *row1*, int *col2*, int *row2*, int **retcol*, int **retrow*, char **match*)**;**

This function waits for text matching *pattern* to appear within the rectangle defined by col1, row1, col2, row2. Text matching the pattern is returned to **match**, and the actual row and column position is returned to **retcol** and **retrow**. If the **pattern** does not appear before the **timeout** expires, the function returns an error code. The **pattern** can include a regular expression. See the *Online Function Reference* for details on using regular expressions. Besides the **pattern** and **timeout** parameters, all the other parameters are optional.

If **pattern** is passed as an empty string, the function will wait for timeout if it finds any text at all within the rectangle. If there is no text, it returns immediately.

If the *pattern* does appear, then the function waits for the emulator to be stable (finish redrawing, and not display any new characters) for the interval defined by the TE_SILENT_SEC and TE_SILENT_MILLI system variables. This, in effect, allows the terminal to become stable and emulates a human user.

If the terminal does not become stable within the interval defined by TE_SILENT_TIMEOUT, script execution continues. The function returns 0 for success, but sets the TE_errno variable to indicate that the terminal was not silent after the text appeared.

To modify or retrieve the value of any of the TE_SILENT system variables, use the **TE_getvar** and **TE_setvar** functions. For more information, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the Vuser types in its name, and then waits for the application to respond.

```
/* Declare variables for TE_wait_text */
int ret_row;
int ret_col;
char ret_text [80];
/* Type in user name. */
TE_type ("John");
/* Wait for teller to respond. */
TE_wait_text ("Enter secret code:", 30, 29, 13, 1, 13, &ret_col, &ret_row,
              ret_text);
```

You can instruct VuGen to automatically generate **TE_wait_text** statements, and insert them into a script, while you record the script.

To instruct VuGen to automatically generate TE_wait_text statements, and insert them into a script while recording:

1 Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.

2 Under **Generate Automatic Synchronization Commands**, select the **Prompt** check box, and then click **OK**.

The following is an example of a TE_wait_text statement that was automatically generated by VuGen. The function waits up to 20 seconds for the string "keys" to appear anywhere on the screen. Note that VuGen omits all the optional parameters when it generates a TE_wait_text function.

```
TE_wait_text("keys", 20);
```

Waiting for the Terminal to be Silent

In instances when neither cursor synchronization nor text synchronization are effective, you can use "silent synchronization" to synchronize the script. With "silent synchronization," the Vuser waits for the terminal emulator to be silent for a specified period of time. The emulator is considered to be silent when it does not receive any input from the server for a specified period of time.

Note: Use silent synchronization only when neither cursor synchronization nor text synchronization are effective.

You use the **TE_wait_silent** function to instruct a script to wait for the terminal to be silent. You specify the period for which the terminal must be silent. If the terminal is silent for the specified period, then the **TE_wait_silent** function assumes that the application has stopped printing text to the terminal screen, and that the screen has stabilized.

The syntax of the function is:

```
int TE_wait_silent (int sec, int milli, int timeout);
```

The **TE_wait_silent** function waits for the terminal emulator to be silent for the time specified by *sec* (seconds) and *milli* (milliseconds). The emulator is considered silent when it does not receive any input from the server. If the emulator does not become silent (i.e. stop receiving characters) during the time specified by the time *timeout* variable, then the function returns an error.

For example, the following statement waits for the screen to be stable for three seconds. If after ten seconds, the screen has not become stable, the function returns an error.

```
TE_wait_silent (3, 0, 10);
```

For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

65

RTE - Reading Text from Terminal Screen

RTE Vusers can read text from the user interface of a terminal emulator, and then perform various tasks with that text.

This chapter includes:

About Reading Text from the Terminal Screen

There are several Vuser functions that RTE Vusers can use to read text from the terminal screen. You can use these functions, **TE_find_text** and **TE_get_text_line**, to check that the terminal emulator is responding correctly, or to enhance the logic in your scripts.

After recording, you can manually insert **TE_find_text** and **TE_get_text_line** statements directly into your RTE Vuser scripts.

Searching for Text on the Screen

The **TE_find_text** function searches for a line of text on the screen. The syntax of the function is:

int **TE_find_text (**char **pattern*, int *col1*, int *row1*, int *col2*, int *row2*,
int **retcol*, int **retrow*, char **match*)**;**

This function searches for text matching *pattern* within the rectangle defined by *col1, row1, col2, row2*. Text matching the pattern is returned to *match*, and the actual row and column position is returned to *retcol* and *retrow*. The search begins in the top-left corner. If more than one string matches *pattern,* the one closest to the top-left corner is returned.

The **pattern** can include a regular expression. See the *Online Function Reference* for details on using regular expressions.

You must manually type **TE_find_text** statements into your Vuser scripts. For details on the syntax of the **TE_find_text** function, see the *Online Function Reference* (**Help** > **Function Reference**).

Reading Text from the Screen

The **TE_get_text_line** function reads a line of text from the area of the screen that you designate. The syntax of the function is:

char ***TE_get_text_line (**int *col*, int *row*, int *width*, char **text*)**;**

This function copies a line of text from the terminal screen to a buffer *text*. The first character in the line is defined by *col*, *row*. The column coordinate of the last character in the line is indicated by *width*. The text from the screen is returned to the buffer *text*. If the line contains tabs or spaces, the equivalent number of spaces is returned.

In addition, the **TE_get_cursor_position** function can be used to retrieve the current position of the cursor on the terminal screen. The **TE_get_line_attribute** function returns the character formatting (for instance, bold or underline) of a line of text.

You must manually type **TE_get_text_line** statements into your Vuser scripts. For details on the syntax of the **TE_get_text_line** function, see the *Online Function Reference* (**Help** > **Function Reference**).

66

Mailing Services Protocols

VuGen allows you to test several mailing services on a protocol level. It emulates the sending of mail, and most of the standard operations performed against a mail server.

This chapter includes:

About Developing Vuser Scripts for Mailing Services

The Mailing Service protocols emulate a user working with an email client, viewing and sending emails. The following mailing services are supported:

- Internet Messaging (IMAP)
- MS Exchange (MAPI)
- Post Office Protocol (POP3)
- Simple Mail Transfer Protocol (SMTP)

The mail protocols support both record and replay, with the exception of MAPI that only supports replay.

When you record an application using one of the mail protocols, VuGen generates functions that emulate the mail client's actions. You can indicate the programming language in which to create a Vuser script —either C or Visual Basic scripting. For more information, see Chapter 76, "Setting Script Generation Preferences." If the communication is performed through multiple protocols, you can record both of them. You can record several mail protocols, or a mail protocol together with HTTP or WinSock. For instructions on specifying multiple protocols, see Chapter 5, "Recording with VuGen."

All Mailing Service functions come in pairs—one for global sessions and one where you can indicate a specific mail session. For example, **imap_logon** logs on to the IMAP server globally, while **imap_logon_ex** logs on to the IMAP server for a specific session.

Getting Started with Mailing Services Vuser Scripts

This section provides an overview of the process of developing Vuser scripts for Mailing Services using VuGen.

To develop a Mailing Service Vuser script:

1 Create a basic script using VuGen.

Invoke VuGen and create a new Vuser script for either a single mail protocol or multiple protocols.

2 Record the basic script using VuGen. (Except MAPI)

Select an application to record. Perform typical operations in your application. For details, see Chapter 5, "Recording with VuGen."

For MAPI, recording is not supported. Instead, you create an empty MAPI script and manually insert **mapi** functions into it. For examples, see the *Online Function Reference* (**Help** > **Function Reference**).

3 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

4 **Define parameters (optional).**

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

5 **Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, "Correlating Statements."

6 **Configure the run-time settings.**

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

7 **Run the script from VuGen.**

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding IMAP Scripts

IMAP Vuser script functions record the Internet Mail Application Protocol. Each IMAP function begins with an **imap** prefix.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

In the following example, the **imap_create** function creates several new mailboxes: Products, Solutions, and FAQs.

```
Actions()
{
    imap_logon("ImapLogon",
            "URL=imap://johnd:letmein@exchange.mycompany.com",
            LAST);

    imap_create("CreateMailboxes",
            "Mailbox=Products",
            "Mailbox=Solutions",
            "Mailbox=FAQs",
            LAST);

    imap_logout();

    return 1;
}
```

Understanding MAPI Scripts

MAPI Vuser script functions record activity to and from an MS Exchange server. Each MAPI function begins with a **mapi** prefix.

Note: To run MAPI scripts, you must define a mail profile on the machine running the script. For example, install Outlook Express, set it as the default mail client, and create a mail account. Alternatively, install Microsoft Outlook, set it as the default mail client, create a mail account and create a mail profile. To create a mail profile in Microsoft Outlook, select **Settings** > **Control Panel** > **Mail** > **Show Profiles** and add a mail profile.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

In the following example, the **mapi_send_mail** function sends a sticky note through an MS Exchange server.

```
Actions()
{
    mapi_logon("Logon",
            "ProfileName=John Smith",
            "ProfilePass=Tiger",
            LAST);
    //Send a Sticky Note message
    mapi_send_mail("SendMail",
            "To=user1@techno.merc-int.com",
            "Cc=user0002t@techno.merc-int.com",
            "Subject=<GROUP>:<VUID> @ <DATE>",
            "Type=Ipm.StickyNote",
            "Body=Please update your profile today.",
            LAST);

    mapi_logout();
    return 1;
}
```

Understanding POP3 Scripts

POP3 Vuser script functions emulate actions using the Post Office Protocol, POP3. Each function begins with a **pop3** prefix.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

In the following example, the **pop3_retrieve** function retrieves five messages from the POP3 server.

```
Actions()
{
pop3_logon("Login", "
          URL=pop3://user0004t:my_pwd@techno.merc-int.com",
          LAST);

// List all messages on the server and receive that value
totalMessages = pop3_list("POP3", LAST);

// Display the received value (It is also displayed by the pop3_list function)
lr_log_message("There are %d messages.\r\n\r\n", totalMessages);

// Retrieve 5 messages on the server without deleting them
pop3_retrieve("POP3", "RetrieveList=1:5", "DeleteMail=false", LAST);
pop3_logoff();
     return 1;
}
```

Understanding SMTP Scripts

SMTP Vuser script functions emulate the Single Mail Transfer Protocol traffic. Each SMTP function begins with an **smtp** prefix.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

In the following example, the **smtp_send_mail** function sends a mail message, through the SMTP mail server, techno.

```
Actions()
{
    smtp_logon("Logon",
            "URL=smtp://user0001t@techno.merc-int.com",
                "CommonName=Smtp Test User 0001",
            NULL);

    smtp_send_mail("SendMail",
                        "To=user0002t@merc-int.com",
                        "Subject=MIC Smtp: Sample Test",
                        "MAILOPTIONS",
                        "X-Priority: 3",
                        "X-MSMail-Priority: Medium",
                        "X-Mailer: Microsoft Outlook Express 5.50.400\r\n",
                        "X-MimeOLE: By Microsoft MimeOLE V5.50.00\r\n",
                        "MAILDATA",
                        "MessageText="
                            "Content-Type: text/plain;\r\n"
                            "\tcharset=\"iso-8859-1\"\r\n"
                            "Test,\r\n"
                            "MessageBlob=16384",
                    NULL);

    smtp_logout();

    return 1;
}
```

67

Tuxedo Protocols

You use VuGen to record communication between a Tuxedo client application and a Tuxedo application server. The resulting script is called a Tuxedo Vuser script.

This chapter includes:

About Tuxedo Vuser Scripts

When you record a Tuxedo application, VuGen generates LRT functions that describe the recorded actions. These functions emulate communication between a Tuxedo client and a server. Each LRT function begins with an **lrt** prefix.

In addition to the **lrt** prefix, certain functions use an additional prefix of **tp**, **tx** or **F**. These sub-prefixes indicate the function type, similar to the actual Tuxedo functions. The **tp** sub-prefix indicates a Tuxedo client tp session. For example, **lrt_tpcall** sends a service request and awaits its reply. The **tx** sub-prefix indicates a global tx session. For example, **lrt_tx_begin** begins a global transaction. The **F** sub-prefix indicates an FML buffer related function. For example, **lrt_Finitialize** initializes an existing buffer.

Functions without an additional prefix emulate standard C functions. For example, **lrt_strcpy** copies a string, similar to the C function **strcpy**.

You can view and edit the recorded script from VuGen's main window. The LRT functions that are recorded during the session are displayed in the VuGen window, allowing you to visually track your network activities.

Before You Record

Before you record, verify that the Tuxedo directory, %TUXDIR%\bin is in the path.

If the environment variables have changed since the last time you restarted VuGen, VuGen may record the original variable value rather than the current value.

To avoid any inconsistencies, you should restart VuGen before recording Tuxedo applications.

Getting Started with Tuxedo Vuser Scripts

This section provides an overview of the process of developing Tuxedo Vuser scripts using VuGen.

To develop a Tuxedo Vuser script:

1 Record the basic script using VuGen.

Invoke VuGen and create a new Vuser script. Specify **Tuxedo 6** (for recording Tuxedo Version 6.x) or **Tuxedo** (for recording Tuxedo Version 7.x) as the type of Vuser. Select an application to record. Record typical operations on your application.

For details, see Chapter 5, "Recording with VuGen."

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, "Correlating Statements."

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding Tuxedo Vuser Scripts

VuGen records typical Tuxedo sessions and generates Tuxedo-specific functions. The functions use an **lrt** prefix. For example, **lrt_tpacall** sends a service request.

The LRT functions are divided into the following categories: Buffer Manipulating, Client/Server Session, Communication, Environment Variable, Error Processing, Transaction Handling, and Correlation functions.

You can also manually program any of the functions into your script. For syntax and examples of the LRT functions, see the *Online Function Reference* (**Help** > **Function Reference**).

After you record a session, VuGen's built-in editor lets you view the recorded code. You can scroll through the script, see Tuxedo statements that were generated by your application, and examine the data that was returned by the server. The VuGen window provides you with valuable information about the recorded Tuxedo session. When you view the script in the main window, you see the sequence in which VuGen recorded your activities.

In the following example, VuGen recorded a client's actions in a Tuxedo bank application. The client performed an action of opening a bank account and specifying all the necessary details. The session was aborted when the client specified a zero opening balance.

```
lrt_abort_on_error();
lr_think_time(65);
tpresult_int = lrt_tpbegin(30, 0);
data_0 = lrt_tpalloc("FML", "", 512);
lrt_Finitialize((FBFR*)data_0);
```

```
/* Fill the data buffer data_0 with new account information */
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=8",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=C",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=MID_INIT", "value=Q", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value=123-456-7890",
        LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
        New York, NY 10000", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=SSN","value=111111111", LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0,"name=LAST_NAME",
"value=Doe",LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0,"name=FIRST_NAME",
"value=BJ",LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT",
"value=0.00",LRT_END_OF_PARMS);

/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, &data_0, &olen_2, 0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();
```

Using Parameters in Tuxedo Scripts

You can define parameters in Tuxedo scripts, as described in Chapter 70, "Working with VuGen Parameters." Note that Tuxedo scripts contain strings of type "name=..." or "value=...". You can only define parameters for the portion of the string following the equal sign (=). For example:

```
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value={parameter_1}",
        LRT_END_OF_PARMS);
```

Note: In general, we recommend that you use **lrt_save_parm** to save a portion of a character array to a parameter. Use **lrt_save_searched_string** when you want to save information, relative to the position of a particular string in a character array. For PeopleSoft Vusers, we recommend that you use **lrt_save_searched_string**, since the reply buffers returned from the PeopleSoft server often differ in size during replay from what was seen during recording.

Running Tuxedo Scripts

If you encounter problems recording or running Tuxedo applications, check that the Tuxedo application runs without VuGen, and that the environment variables have been defined correctly. For more information, see "Viewing Tuxedo Buffer Data" below. Note that after you set or modify the Tuxedo variables, you should restart VuGen and your application, in order for the changes to take effect. If your application is 16-bit, then you also need to kill the NTVDM process.

If you experience problems during execution, check the Tuxedo log file on the side of the server for error messages. By default, this file is found in the directory indicated by the environment variable APPDIR. The file name has the form ULOG.mmddyy, where mmddyy indicates the current month, day, and year. The file for March 12, 1999 would be ULOG.031299. The default location of this file can be changed by setting the environment variable ULOGPFX on the server. A log file can also be found on the client side, in the current directory, unless the ULOGPFX variable changes its location.

Viewing Tuxedo Buffer Data

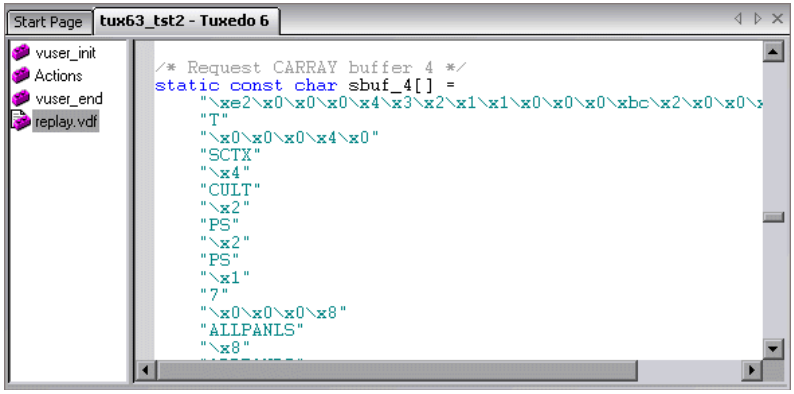
When you use VuGen to create a Tuxedo Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**.

The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file is called **replay.vdf**, and it contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRT functions use the buffer descriptors to access the data.

You can use VuGen to view the contents of the data file by selecting the **replay.vdf** file in the left pane's tree view.

The option to view a data file is available by default for Tuxedo scripts.



Defining Environment Settings for Tuxedo Vusers

The following section describes the system variable settings for Tuxedo Vusers running on Windows and UNIX platforms. You define the system variables in your Control Panel/System dialog box (NT) or .cshrc or .login file (UNIX).

TUXDIR	the root directory for Tuxedo sources.
FLDTBLDIR	list of directories containing FML buffer information. In Windows, separate the names of directories with semi-colons. On UNIX platforms, separate the names of the directories with a colon.
FIELDTBLS	list of files containing FML buffer information. On both Windows and UNIX platforms, separate the file names with commas.

For example:

```
SET FLDTBLDIR=%TUXDIR%\udataobj;%TUXDIR%\APPS\WS (PC)
SET FIELDTBLS=bankflds,usysflds (PC)
setenv FLDTBLDIR $TUXDIR/udataobj:$TUXDIR/apps/bankapp (Unix)
setenv FIELDTBLS bank.flds,Usysflds (Unix)
```

You must define the following system variables for Tuxedo clients using Tuxedo/WS workstation extensions during execution:

WSNADDR	specifies the network address of the workstation listener process. This enables the client application to access Tuxedo. Note that to define multiple addresses in a WSNADDR statement, each address must be separated by a comma.
WSDEVICE	specifies the device that accesses the network. Note that you do not need to define this variable for some network protocols.

For example:

```
SET WSNADDR=0x0002ffffc7cb4e4a (PC)
setenv WSNADDR 0x0002ffffc7cb4e4a (Unix)
setenv WSDEVICE /dev/tcp (Unix)
```

Debugging Tuxedo Applications

In general, use **Tuxedo 6** to record applications using Tuxedo 6.x or earlier, and use **Tuxedo** to record applications using Tuxedo 7.1 and higher.

If you encounter problems recording or replaying Tuxedo applications, or the script is missing a call to lrt_tpinitialize, contact Customer Support to check which DLLs are used with the application.

If the application uses **wtuxws32.dll**, instead of **libwsc.dll**, contact Customer Support to obtain a patch to enable the recording.

Correlating Tuxedo Scripts

VuGen supports correlation for Vuser scripts recorded with Tuxedo applications. Correlated statements enable you to link statements by saving a portion of a buffer and use it in subsequent statements.

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

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

For additional information about the syntax of these functions, see the *Online Function Reference*.

Correlating FML and FML32 Buffers

Use **lrt_save_fld_val** or **lrt_save32_fld_val** to save the contents of the FML or FML32 buffer.

To correlate statements using lrt_save_fld_val:

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

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

2 Reference the parameter.

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

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

```
/* Fill the data_0 buffer with new account information*/
data_0 = lrt_tpalloc("FML", "", 512);
lrt_Finitialize((FBFR*)data_0);
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=1",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=S",
LRT_END_OF_PARMS);
…

LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=LAST_NAME", "value=Doe", …);
lrt_Fadd_fld((FBFR*)data_0, "name=FIRST_NAME", "value=John", …);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=234.12", …);

/* Open a new account and save the new account number*/
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0,&data_0, &olen_2, 0);
lrt_abort_on_error();
lrt_save_fld_val((FBFR*)data_0, "name=ACCOUNT_ID", 0, "account_id");

/* Use result from first query to fill buffer for the deposit*/
lrt_Finitialize((FBFR*)data_0);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCOUNT_ID", "value={account_id}",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=200.11", …);
```

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

You can correlate by field ID as follows:

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

Correlating Character Strings

Use **lrt_save_parm** or **lrt_save_searched_string** to correlate character strings.

- In general, we recommend that you use **lrt_save_parm** to save a portion of a character array to a parameter.
- Use **lrt_save_searched_string** when you want to save information, relative to the position of a particular string in a character array. If the Vuser is for PeopleSoft, we recommend that you use **lrt_save_searched_string**, since the reply buffers returned from the PeopleSoft server often differ in size during replay from what was seen during recording.

Determining Which Values to Correlate

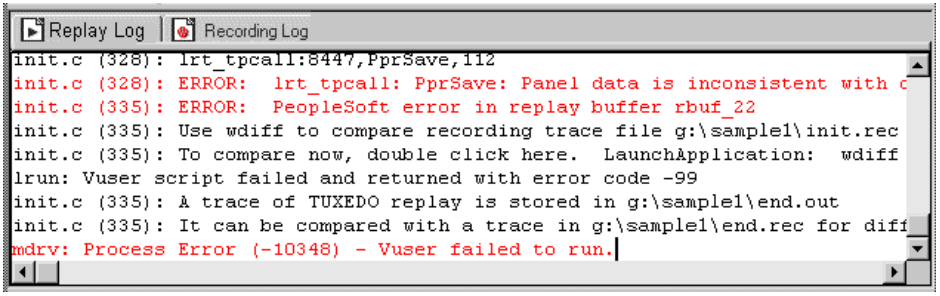
When working with CARRAY buffers, VuGen generates log files during recording (with the **.rec** extension) and during replay (with the **.out** extension) which you can compare using the Wdiff utility. You can look at the differences between the recording and replay logs to determine which portions of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

To compare the log files:

1 Select **View** > **Output** to display the execution log and recording log for your script.

2 Examine the Replay Log tab.

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



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

WDiff opens and the differences between the record and replay files are highlighted in yellow. For more details about the Wdiff utility, see Chapter 8, "Correlating Statements."

To correlate statements using lrt_save_parm:

Once you decide which value to correlate, you can use **lrt_save_parm** to save a portion of a character array (such as a STRING or CARRAY buffer) to a parameter.

1 Insert the **lrt_save_parm** statement in your script at the point where you want to save the contents of the current buffer.

lrt_save_parm (buffer, offset, length, "**param_name**");

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

View the buffer contents by selecting the **replay.vdf** file in the Data Files box of the main VuGen window.

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

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

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

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

```
/* Request CARRAY buffer 57 */
     lrt_memcpy(data_0, buf_143, 227);
     tpresult_int = lrt_tpcall("PprLoad",
          data_0, 227, &data_1, &olen, TPSIGRSTRT);
     lrt_save_parm(data_1, 123, 9, "empid");
```

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

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

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

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

To correlate statements using lrt_save_searched_string:

Use **lrt_save_searched_string** to search for a string in a buffer, and save a portion of the buffer, relative to the string occurrence, to a parameter.

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

```
lrt_save_searched_string (buffer, buf_size, occurrence, string, offset,
                                    length, "param_name");
```

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

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

View the buffer contents by selecting the **replay.vdf** file in the Data Files box of the main VuGen window.

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

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

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

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

```
/* Request CARRAY buffer 1 */
lrt_memcpy(data_0, sbuf_1, 41);
lrt_display_buffer("sbuf_1", data_0, 41, 41);
data_1 = lrt_tpalloc("CARRAY", "", 8192);
tpresult_int = lrt_tpcall("GetCertificate",
     data_0,
     41,
     &data_1,
     &olen,
     TPSIGRSTRT);

/* Reply CARRAY buffer 1 */
lrt_display_buffer("rbuf_1", data_1, olen, 51);
lrt_abort_on_error();

lrt_save_searched_string(data_1, olen, 0, "SCertRep", 9, 16, "cert1");
```

68

Real and Media Player Protocols

Streaming media is a rapidly growing market that allows for the delivery of audio/visual content over the Internet. The idea behind streaming media is that the audio/video content can be transmitted to the end user without having to first download the file in its entirety. Streaming works by having the server continuously stream the content to the client as it displays it.

RealPlayer is an application that display streaming content.

You use VuGen to record communication between a client application and a server that communicate using the RealPlayer protocol. The resulting script is called a Real Vuser script.

This chapter includes:

Note: The Media Player (MMS) protocol should not be confused with the Multimedia Messaging Service (MMS) protocol. For information, see "MMS (Multimedia Messaging Service) Vuser Scripts" on page 1051

About Recording Streaming Data Virtual User Scripts

The Streaming Data protocols allows you to emulate a user playing media or streaming data files.

When you record an application using a streaming data protocol, VuGen generates functions that describe your actions. For RealPlayer sessions, VuGen generates functions with an **lreal** prefix. For Media Player sessions, VuGen uses functions with an **mms** prefix. Note that recording is not supported for Media Player mms functions—only replay.

Getting Started with Streaming Data Vuser Scripts

This section provides an overview of the process of developing RealPlayer and Media Player streaming data Vuser scripts using VuGen.

To develop a Real or Media Player Vuser script:

1 Record the basic script using VuGen. (Real only)

Invoke VuGen and create a new Vuser script. Select an application to record, and record typical operations on your application. For details, see Chapter 5, "Recording with VuGen."

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 70, "Working with VuGen Parameters."

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, "Correlating Statements."

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 79, "Configuring Run-Time Settings."

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using RealPlayer LREAL Functions

The functions developed to emulate communication between a client and a server by using the RealPlayer protocol are called Real Player functions. Each Real Player function has an **lreal** prefix.

VuGen automatically records most of the LREAL functions listed in this section during a Real Player session. You can also manually program any of the functions into your script.

For example, the **lreal_play** function takes the form:

```
int lreal_play (int miplayerID, long mulTimeToPlay);
```

To play the clip until the end, use any negative value for **mulTimeToPlay**. To play the clip for a specific duration number of milliseconds, specify the number of milliseconds. **miplayerID** represents a unique ID of a RealPlayer instance.

For more information about the LREAL functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Using Media Player MMS Functions

The functions developed to emulate client/server communication for Media Player's MMS protocol, are called MMS Virtual User functions—each function has an **mms** prefix.

Important: In order to replay Media Player functions, you must place a file called **wmload.asf** on the Windows Media server machine. The VuGen machine must be able to access using **mms://<servername>/wmload.asf**. This ASF file can be any media file renamed to **wmload.asf**.

All MMS functions come in pairs—one for global sessions and one for a specific session. For example, **mms_close** closes the Media Player globally, while **mms_close_ex** closes the Media Player for a specific session.

A typical function, **mms_play**, takes the following form:

```
int mms_play (char message, <List of Attributes>, LAST);
```

In the following example, the **mms_play** function plays an **asf** file for different durations:

```
//Play for a duration of 10 seconds.
mms_play("Welcome","URL=mms://server/welcome.asf","
duration=10",
LAST);

//Play the clip until its completion, after waiting 5 seconds.
mms_play ("Welcome","URL=mms://server/welcome.asf",
"duration=-1",
"starttime=5",
LAST);
```

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

69

Wireless Protocols

VuGen enables you to generate Wireless Vuser scripts by recording typical Wireless sessions. When you run a script, the resulting Vuser emulates activity between your toolkit or phone and Web server (or gateway for WAP).

This chapter includes:

Understanding the WAP Protocol

The Wireless Application Protocol (WAP) is an open, global specification that enables mobile users with wireless devices to instantly access and interact with information and services.

The WAP protocol specifies a microbrowser thin-client using a new standard called WML that is optimized for wireless handheld mobile terminals. WML is a stripped-down version of XML.

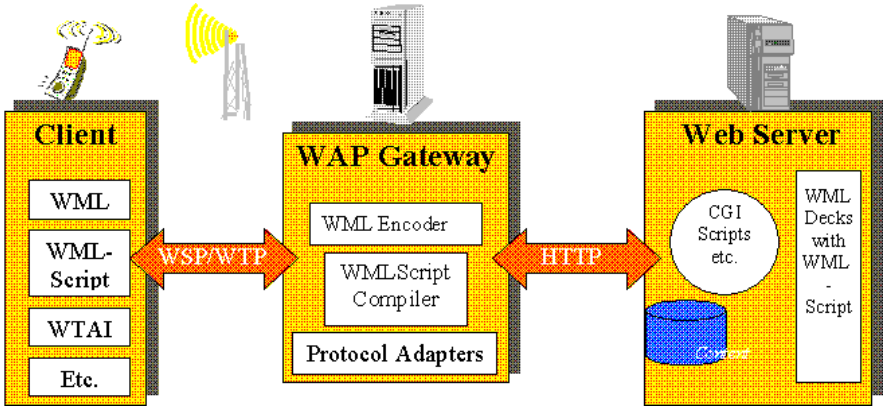
WAP also specifies a proxy server that:

- acts as a gateway between the wireless network and the wire-line Internet
- provides protocol translation
- optimizes data transfer for the wireless handset

WAP architecture closely resembles the WWW model. All content is specified in formats that are similar to the standard Internet formats. Content is transported using standard protocols in the WWW domain and an optimized HTTP-like protocol in the wireless domain (Wireless Session Protocol). You locate all WAP content using WWW standard URLs.

WAP uses many WWW standards, including authoring and publishing methods. WAP enhances some of the WWW standards in ways that reflect the device and network characteristics. WAP extensions are added to support Mobile Network Services such as Call Control and Messaging. It accounts for the memory and CPU processing constraints that are found in mobile terminals. WAP also supports low bandwidth and high latency networks.

WAP assumes the existence of a gateway that is responsible for encoding and decoding data transferred to and from the mobile client. The purpose of encoding content delivered to the client is to minimize the size of data sent to the client over-the-air, as well as to minimize the computational energy required by the client to process that data. The gateway functionality can be added to origin servers, or placed in dedicated gateways as illustrated below.



WAP Toolkits

To assist developers in producing WAP applications and services, the leading companies such as Nokia, Ericsson, and Phone.com, have developed toolkits. The WAP Toolkit provides an environment for developers who want to provide Internet services and content for mobile terminals. It allows developers to write, test, debug, and run applications on a PC-based simulator phone. The toolkit allows users to browse WAP sites through an HTTP connection or a WAP gateway.

A mobile phone communicates with a gateway in WSP protocol; a toolkit can communicate with the gateway, or directly with the server. VuGen automatically detects the communication mode that is configured in the toolkit: WSP or HTTP. If you are interested in the traffic to the gateway, you record in WSP mode. If you want to check the server and the content providers, you can record your toolkit session in HTTP mode, and bypass the gateway.

VuGen uses custom API functions to emulate a user session. Most functions are the standard Web protocol functions utilizing the HTTP protocol. Several WAP functions emulate actions specific to WAP Vusers. For a list of the supported functions, see "Using Wireless Vuser Functions" on page 1047.

Getting Started with Wireless Vuser Scripts

Wireless Vuser Scripts emulate a user using a wireless browser. You record the user browsing on a PC-based simulator phone (toolkit). You can then distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

This section provides an overview of the process of developing Wireless Vuser scripts using VuGen.

To develop a Wireless script:

1 Create a new script using VuGen.

Select **File** > **New** or click the **New** button to create a new script in either single or multiple protocol mode.

For details about creating a new script, see Chapter 5, "Recording with VuGen."

2 Record the actions using VuGen.

Record the actions over the toolkit session. VuGen automatically detects the toolkit settings and uses those settings during recording.

For information about recording, see Chapter 5, "Recording with VuGen."

3 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, "Enhancing Vuser Scripts."

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values. For details, see Chapter 70, "Working with VuGen Parameters."

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include the run logic, pacing, logging, think time, performance preferences, and gateway settings.

For information about the General run-time settings, see Chapter 79, "Configuring Run-Time Settings."

For information about common Internet protocol run-time settings, see Chapter 80, "Configuring Network Run-Time Settings".

For information about WAP specific run-time settings, see "WAP Run-Time Settings" on page 1330.

6 Perform correlation.

Check your script to determine if there are dynamic values that require correlation. For Wireless protocols, you perform manual correlation by adding **web_reg_save_param** functions.

For more information, see "Performing Manual Correlation" on page 875.

7 Save and run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly. 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.

For details about running the Vuser script as a standalone test, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using Wireless Vuser Functions

The functions developed to emulate communication between a wireless instrument and Web server (or gateway for WAP), are called Vuser functions. Some functions are generated when you record a script; others you must manually insert into the script. You can also add Vuser message functions and custom C functions to your Vuser scripts after recording.

General Vuser functions begin with an **lr** prefix. The functions representing standard HTTP actions, have a **web** prefix.

Functions that are specific to WAP, have a **wap** prefix. For example, **wap_connect** connects to a WAP gateway. WAP functions are divided into the following categories: Bearer, Connection, Formatting, Gateway, Push, and Radius functions.

Functions that emulate a RAS or NAS server during a WAP session, have a **radius** prefix. For example, **radius_account** authenticates a user to a RADIUS server.

For a complete list of all VuGen functions, see the *Online Function Reference* (**Help > Function Reference**).

For WAP Vusers running scripts in Wireless Session Protocol (WSP) mode, only the following functions are supported:

Action Functions	**web_custom_request**, **web_submit_data**, and **web_url**
Authentication Functions	All—**web_set_user**, **web_set_certificate**[_ex]
Cookie Functions	All—**web_add_cookie**, **web_cleanup_cookie**, **web_remove_cookie**
Header Functions	All —**web_add_auto_header**, **web_add_header**, **web_cleanup_auto_headers**, **web_save_header**
Correlation Functions	All—**web_create_html_param**[_ex], **web_reg_save_param**, **web_set_max_html_param_len**

Push Support

In the normal client/server model, a client requests information or a service from a server. The server responds by transmitting information or performing a service to the client. This is known as **pull** technology—the client pulls information from the server.

In contrast to this, there is also **push** technology. The WAP push framework transmits information to a device without a previous user action. This technology is also based on the client/server model, but there is no explicit request from the client before the server transmits its content.

To perform a push operation in WAP, a **Push Initiator** (PI) transmits content to a client. However, the Push Initiator protocol is not fully compatible with the WAP Client—the Push Initiator is on the Internet, and the WAP Client is in the WAP domain. Therefore, we need to insert a translating gateway to serve as an intermediary between the Push Initiator and the WAP Client. The translating gateway is known as the **Push Proxy Gateway** (PPG).

The access protocol on the Internet side is called the **Push Access Protocol** (PAP).

The protocol on the WAP end is called the Push **Over-The-Air** (OTA) protocol.

The Push Initiator contacts the Push Proxy Gateway (PPG) over the Internet using the PAP Internet protocol. PAP uses XML messages that may be tunneled through various well-known Internet protocols such as HTTP. The PPG forwards the pushed content to the WAP domain. The content is then transmitted using the OTA protocol over the mobile network to the destination client. The OTA protocol is based on WSP services.

In addition to providing basic proxy gateway services, the PPG is capable of notifying the Push Initiator about the final status of the push operation. In two-way mobile networks, it can also wait for the client to accept or reject the content.

Push Services Types

Push services can be of the SL or SI type:

- **SL.** The Service Loading (SL) content type provides the ability to cause a user agent on a mobile client to load and execute a service—for example, a WML deck. The SL contains a URI indicating the service to be loaded by the user agent without user intervention when appropriate.
- **SI.** The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. For example, the notifications may be about new emails, changes in stock price, news headlines, and advertising.

In its most basic form, an SI contains a short message and a URI indicating a service. The message is presented to the end-user upon reception, and the user is given the choice to either start the service indicated by the URI immediately, or postpone the SI for later handling. If the SI is postponed, the client stores it and the end-user is given the ability to act upon it at a later point of time.

VuGen Push Support

Push support for VuGen is divided into three parts:

- Push support at the client end—the ability to accept push messages.
- Push support to WAP HTTP Vusers—emulating Push Initiators.
- Push messages (SI & SL) format services—formatting push messages.

Client Push Support

At the client end, VuGen supports both push services (SL and SI) for all replay modes (CO and CL). The **wap_wait_for_push** function instructs the Vuser to wait for a push message to arrive. You set the timeout for this function in the run-time settings.

When a push message arrives, the Vuser parses it to determine its type and to retrieve its attributes. If parsing was successful, it generates and executes a pull transaction to retrieve the relevant data. You can disable the pull event, indicating to the Vuser not to retrieve the message data by configuring the Run-Time settings. For more information, see "WAP Run-Time Settings" on page 1330.

Emulating a Push Initiator

Push support for WAP HTTP Vusers enables you to perform load testing of the PPG. Push support allows Vusers to function as Push Initiators supporting the **Push Access Protocol** (PAP). The PAP defines the following sets of operations between the PI and the PPG:

- Submit a Push request.
- Cancel a Push request.
- Submit a query for the status of a push request.
- Submit a query for the status of a wireless device's capabilities.
- Initiate a result notification message from the PPG to the PI.

All operations are request/response—for every initiated message, a response is issued back to the PI. PI operations are based on the regular HTTP POST method supported by VuGen. Currently, only the first two operations are supported through **wap_push_submit** and **wap_push_cancel.**

You can submit data to a Web server using the **web_submit_data** function. It is difficult, however, to send long and complex data structures using this function. To overcome this difficulty and provide a more intuitive API function, several new API functions were added to properly format the XML message data: **wap_format_si_msg** and **wap_format_sl_msg**. For more information about these functions, see the *Online Function Reference*.

MMS (Multimedia Messaging Service) Vuser Scripts

MMS (Multimedia Messaging Service) is an extension of the SMS protocol. Whereas SMS messages can only contain text, MMS allows you to send and receive messages with a wide range of content to MMS capable handsets. This content can be in the form of text, sound, email messages, images, video clips, and even streaming data. It is also possible to send multimedia messages from a mobile phone to an email address.

An MMS message typically includes a collection of attachments. While SMS messages are limited to 160 bytes, an MMS message could be several MBs in size. MMS usually requires a third generation (3G) network to enable such large messages to be delivered.

To receive an MMS message, a mobile phone receives an MMS notification over SMS. The SMS message can be received over various SMS protocols such as SMPP, UCP, and CIMD2. The SMS message contains a unique path to the MMS message stored in the MMSC server's database and the mobile phone uses this path to download the message from the SMSC. The current version of VuGen supports the receiving of MMS notifications over the SMPP interface.

Multimedia Messaging Service Vuser scripts support the 1.0 and 1.1 versions of the MMS protocol, as defined by OMA (Open Mobile Alliance organization). Using MMS Vusers, you can send MMS messages to the MMSC server directly over the HTTP protocol, or over the WAP protocol through a WAP gateway.

Multimedia Messaging Service functions emulate the sending and receiving of MMS messages. Each function begins with an **mm** prefix. For detailed syntax information for these functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Running an MMS Scenario in the Controller

An MMS (Multimedia Messaging Service) scenario requires a command line setting.

To set the MMS command line setting:

1. From the Scenario Schedule screen, click **Details**. The Group Information dialog is displayed.
2. If the Command line box is not visible, click the **More** button.
3. Add the following to the end of the Command line text: -usingwininet yes
4. Click **OK** to accept the Command line switch.

Part 3

Parameters

70

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.

This chapter includes:

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 1061.

Parameterizing a Vuser script has the following 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 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, see the *Online Function Reference* (**Help** > **Function Reference**) for each function.

For example, consider the lrd_stmt function. The function has the following syntax:

```
lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long mliTextLen,
LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int miDBErrorSeverity);
```

The *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, see 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:

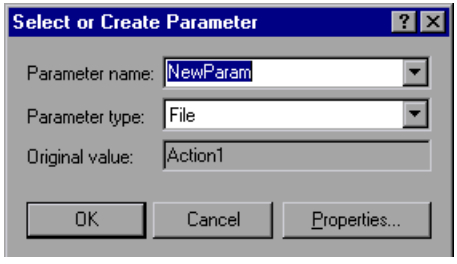
- When creating XML parameters in script view, you must select only the inner xml, without the bounding tags. For example, to parameterize the complex data structure <A><B>Belement</B><C>Celement</C></A>, select the whole string, <B>Belement</B><C>Celement</C>, and replace it with a parameter.
- When parameterizing Java Record Replay or 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 1061.

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 1064.

Step 5: Replace the argument with the parameter.

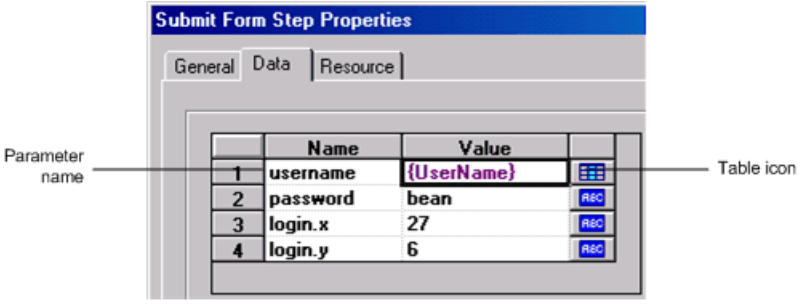
Click **OK** to close the Select or Create Parameter dialog box.

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 check the proper parameter braces in the Parameterization tab (select **Tools** > **General Options**). For more information, see "Setting Parameterization Options" on page 1071.

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

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 71, "File, Table, 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. We recommend that you use XML parameters with Web Service scripts or for SOA services.

Internal Data Parameter Types

Internal data is generated automatically while a Vuser runs, such as Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.

For information about defining Internal Data parameter properties see "Setting Properties for Internal Data Parameter Types" on page 1102.

User-Defined Function Parameters

Data that is generated using a function from an external DLL. A user-defined function replaces the parameter with a value returned from a function located in an external DLL.

Before you assign a user-defined function as a parameter, you create the external library (DLL) with the function. The function should have the following format:

```
__declspec(dllexport) char *<functionName>(char *, char *)
```

The arguments sent to this function are both NULL.

When you create the library, we recommend that you use the default dynamic library path. That way, you do not have to enter a full path name for the library, but rather, just the library name. VuGen's bin 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 1112.

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:

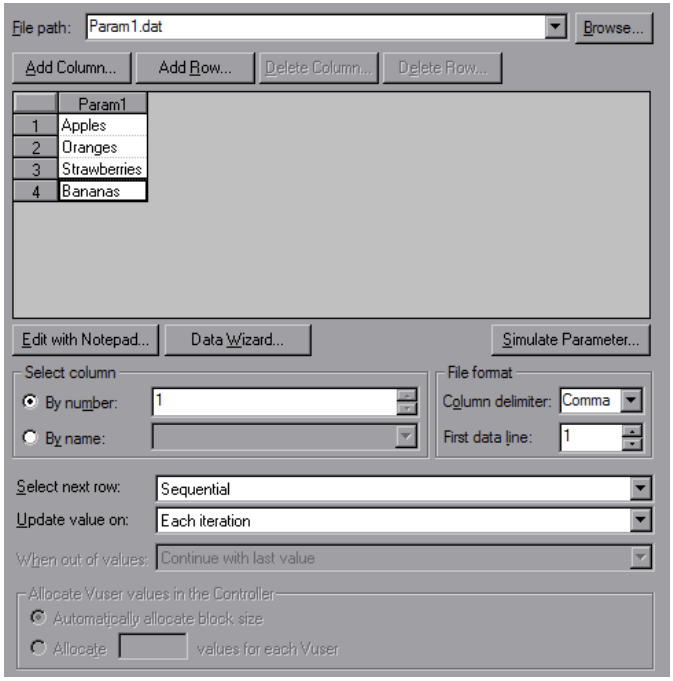
1 Open the Parameter Properties dialog box.

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 1058, 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 select **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 71, "File, Table, and XML Parameter Types."
- To define properties for internal data parameter types, see "Setting Properties for Internal Data Parameter Types" on page 1102.
- To define properties for user-defined functions, see "User-Defined Function Parameters" on page 1063.

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 1068.

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.

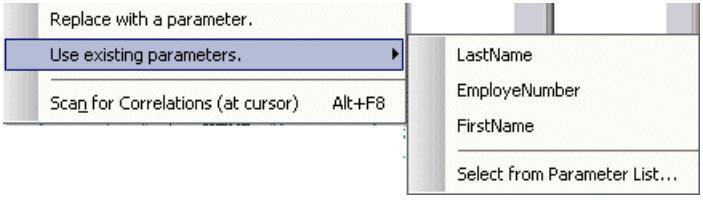
Replacing Strings Using Pre-defined Parameters

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.
- Select **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 1068.

Replacing Multiple Occurrences

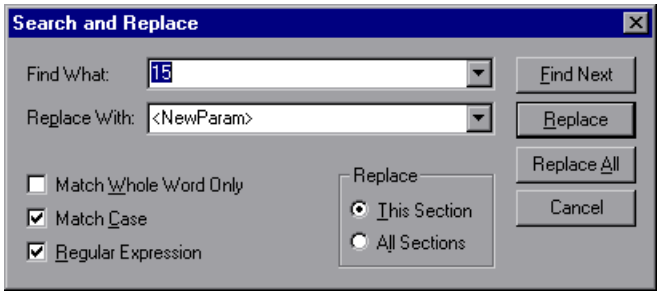
When you create a parameter, the system remembers the original value of the argument. You can use the **Search and Replace** function to replace selected or all occurrences of the same argument with the same parameter or another existing parameter.

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

1 Right-click a parameter and select **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 view, create, delete, select, and modify parameters.

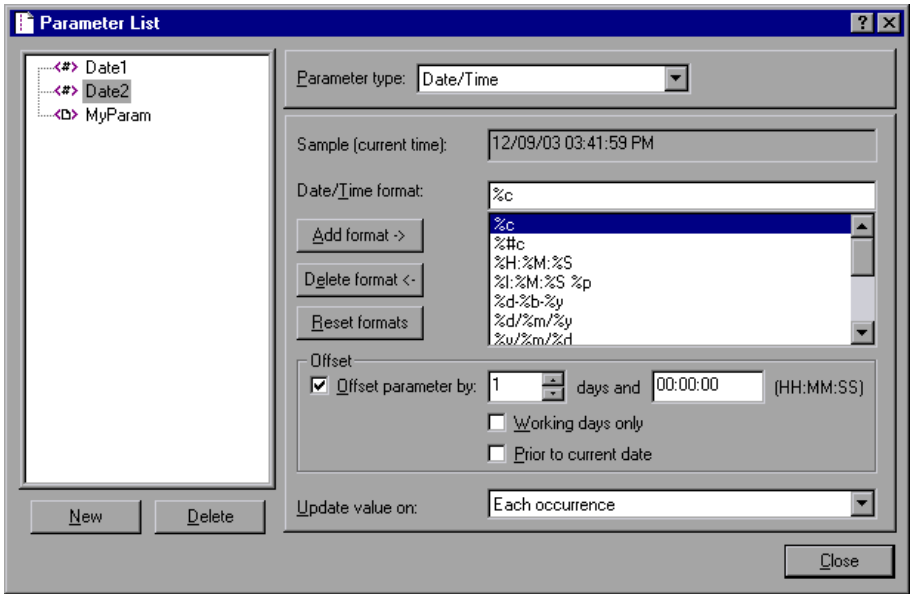
To view the Parameter List and view a parameter's properties:

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

The Parameter list shows all of the parameters that you created, including both input and output parameters. Input parameters are parameters whose value you define in the design stage before running the script. Output parameters you define during design stage, but they acquire values during test execution. Output parameters are often used with Web Service calls.

Use care when selecting a parameter for your script during design stage, make sure that it is not an empty Output parameter.

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 72, "Setting Parameter Properties," and Chapter 71, "File, Table, 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 **Parameterization** tab (**Tools** > **General Options**) shows the default braces for your protocol. In the following example, the Web protocol uses curly brackets:

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

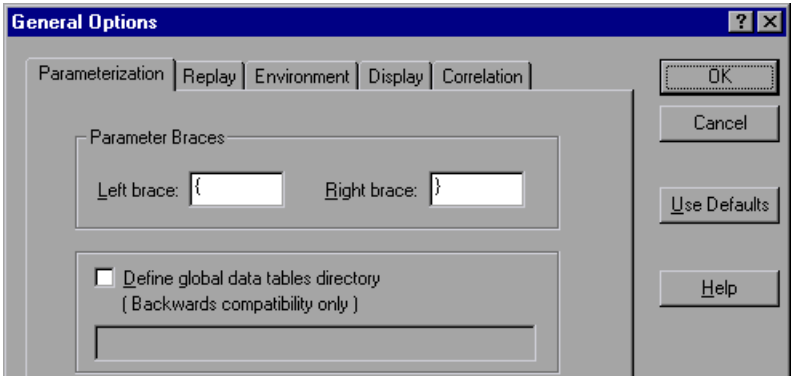
You can change the style of parameter braces by specifying a string of one or more characters. All characters are valid with the exception of spaces.

Note: The default parameter braces are either angle or curly brackets, depending on the protocol type. To check the default brace type, create a new script and check the **Parameterization** tab (select **Tools** > **General Options**).

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 1062.

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.

71

File, Table, and XML Parameter Types

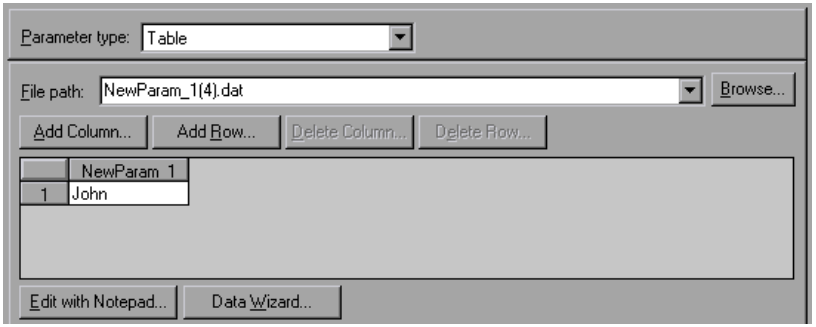
A very common method for using parameters is instructing Vusers to take values from a data table or an external file. The data is contained either in an existing file or in a file that you create with VuGen or MS Query.

This chapter includes:

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:

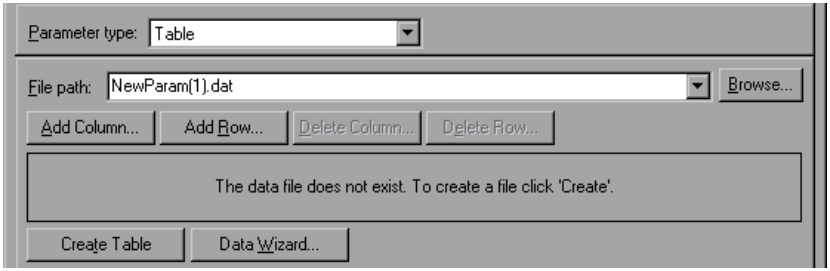
1 Open the Parameter Properties dialog box or the Parameter List.

For instructions, see "Defining Parameter Properties" on page 1064.

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 select 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.



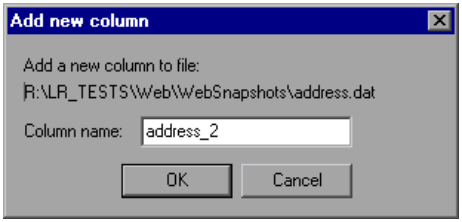
VuGen opens the data file and displays the first 100 rows. To view all of the data, click **Edit with Notepad** and view the data in a text editor.

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 1072.

➤ 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 1078.

3 Add columns and rows to the table.

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

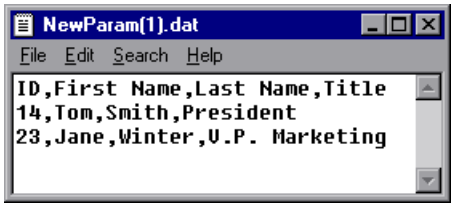


➤ To add additional rows to the table, select **Add Row**.

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

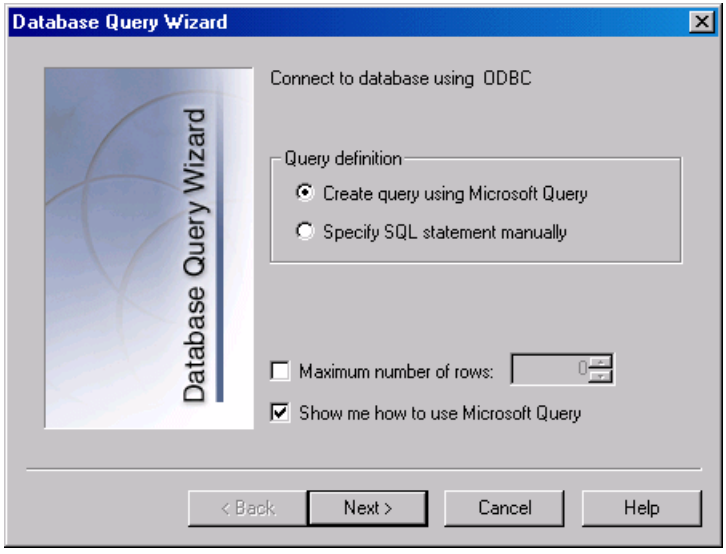
VuGen allows you to import data from a database for use with parameterization. You can import the data in one of two ways:

➤ 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.



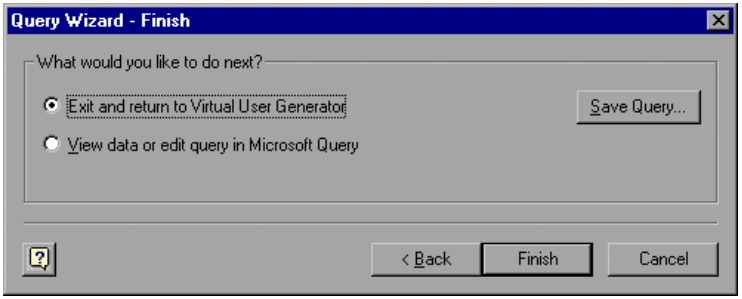
Creating a New Query

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



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

5 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 1082.

Specifying an SQL Statement

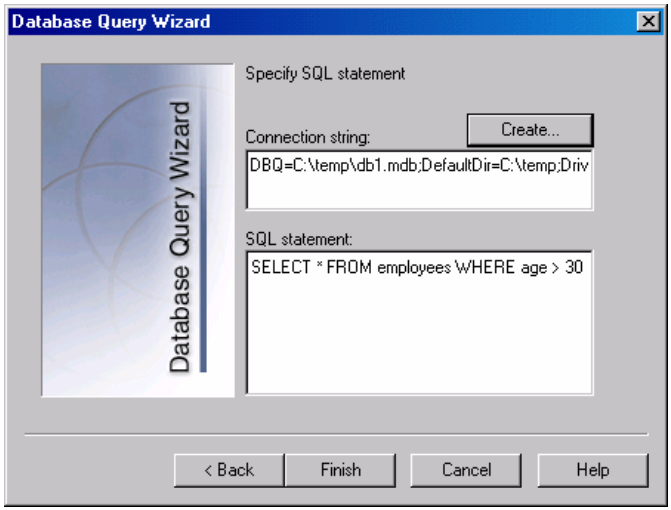
To specify a database connection and SQL statement:

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 statement** box, enter an SQL statement.



5 Click **Finish** to process the SQL statement and import the data. The database records appears in the Parameter Properties box as a data file.

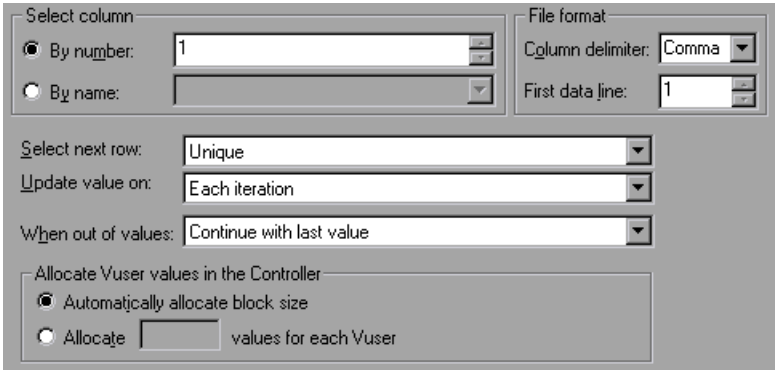
6 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 1082.

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 1068.

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 select 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 1086.

5 Select 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 1088.

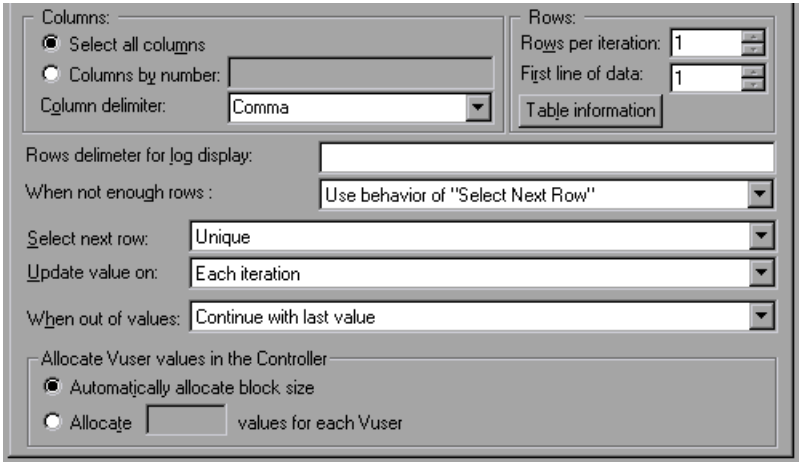
6 If you chose **Unique** as the Data Assignment method (in step 4):

- **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. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate.

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

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. Alternatively, you can select **Select all columns.**

To specify one or more columns by their number, select **Columns by number** and enter the column numbers separated by a comma or a dash. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, select 1.

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. Select **Parameter will get less rows than required** to fill in only two rows. Select **Use behavior of "Select Next Row"** to loop around and get the next row according the method specified in the **Select next row** box—**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 1086.

8 Select 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 1088.

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. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate.

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

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 see 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 1082, or "Setting Properties for Table Type Parameters" on page 1084.

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

Sequential Method

- If you specify update on **Each iteration**, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, and so on.
- If you specify update on **Each occurrence**, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, and so on.
- If you specify update **Once**, all Vusers take Kim for all iterations.

Note: If you select the **Sequential** method and there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random Method

- If you specify update on **Each iteration**, the Vusers use random values from the table for each iteration.
- If you specify update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
- If you specify update **Once**, all Vusers take the first randomly assigned value for all the iterations.

Unique Method

- If you specify update on **Each iteration**, for a test run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane, Ron, and Alice. The third Vuser, Ken, Julie, and Fred.
- If you specify update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.

➤ If you specify update **Once**, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, and so on.

Vuser Behavior in the Controller (LoadRunner Only)

When you set up a scenario to run a parameterized script, you can instruct the Vusers how to act when there are not enough values. The following table summarizes the results of a scenario using the following parameter settings:

➤ Select next row = **Unique**

➤ Update Value on = **Each iteration**

➤ When out of values = **Continue with last value**

Situation	Duration	Resulting Action
More iterations than values	**Run until completion**	When the unique values are finished, each Vuser continues with the last value, but a warning message is sent to the log indicating that the values are no longer unique.
More Vusers than values	**Run indefinitely or Run for …**	Vusers take all of the unique values until they are finished. Then the test issues an error message **Error: Insufficient records for param <param_name> in table to provide the Vuser with unique data**. To avoid this, change the **When out of values** option in the Parameter properties or the **Select next row** method in the Parameter properties.
One of two parameters are out of values	**Run indefinitely or Run for …**	The parameter that ran out of values, continues in a cyclic manner until the values of the second parameter are no longer unique.

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 elements and assign a different value set for each iteration.

The XML parameter type supports complex schema types such as arrays. Choice, and <any> elements.

This section describes:

- Creating New XML Parameters
- Defining Value Sets
- Setting an Assignment Method
- Modifying XML Parameter Properties

Creating New XML Parameters

When working with Web Service **Input Arguments**, you may encounter arrays and their sub-elements. You can define a single XML parameter that will contain values for all of the array elements.

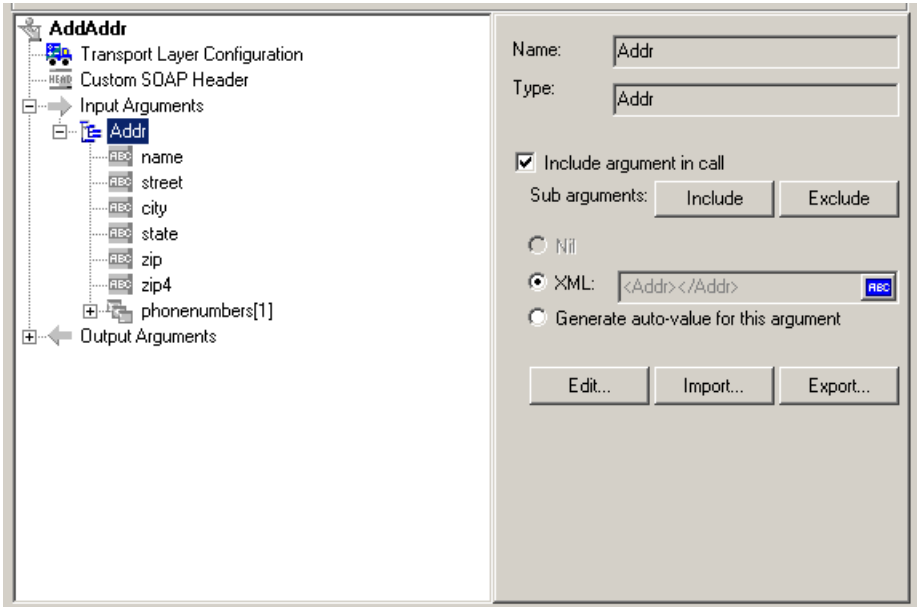
You can create new XML type parameters directly from the **Insert** menu, similar to all other parameter types. For Web Services type scripts, you create an XML parameter directly from the Web Services Call properties.

Creating XML Parameters From a Web Service Call

This section describes how to create an XML parameter from the Web Service properties.

To create an XML parameter from the Web Service call properties:

1 Select the root element of the complex data structure. The right pane displays the argument's details.



2 Select **XML** in the right pane, and click the **ABC** icon. The Select or Create Parameter dialog box opens.

3 In the **Parameter name** box, enter a name for the parameter.

4 In the **Parameter type** box, select **XML** if it is not already selected.

5 Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

Creating XML Parameters - Standard Method

This section describes how to create an XML type parameter without viewing the properties of a Web Service call. This is the most common way of parameterizing values for most protocols and parameter types.

For Web Service Scripts, we recommend that you create parameters from within a Web Service Call, as described above.

To create a new XML parameter:

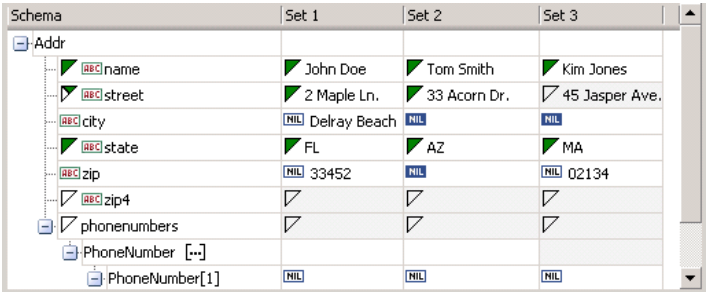
1. Select **Insert** > **New Parameter** or select a constant value in the Script view and select **Replace with a Parameter** from the right-click menu. The Select or Create Parameter dialog box opens.
2. In the **Parameter name** box, enter a name for the parameter.
3. In the **Parameter type** box, select **XML** if it is not already selected.
4. Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

For information on how to set the properties, see "Setting Properties for XML Parameters" on page 1091.

Defining Value Sets

This section describes how to create value sets for XML parameters.

Value sets are arrays that contain a set of values. Using the **Add Column** and **Duplicate Column** buttons, you can create multiple value sets for your parameter and use them for different iterations.



When using value sets, the number of array elements per parameter does not have to be constant.

You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

To exclude an optional element, click the small triangle in the upper left corner of the cell and insure that it is not filled in.

In the following example, **Set 1** and **Set 2** use the optional elements: **name**, **street**, and **state**. **Set 3** does not use a street name.

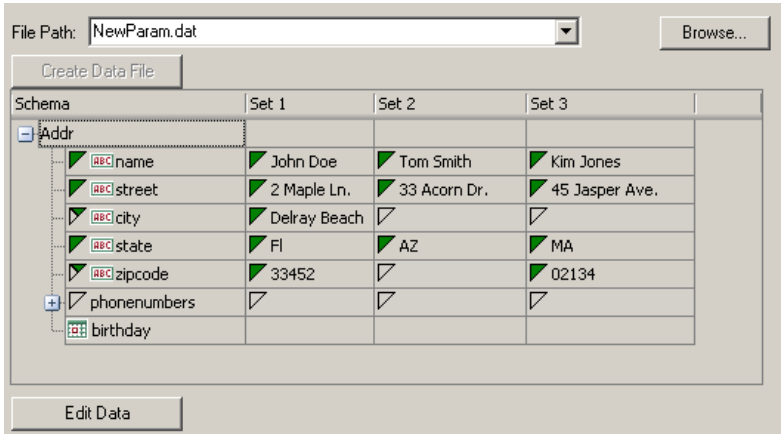
Schema	Set 1	Set 2	Set 3
Addr			
name	John Doe	Tom Smith	Kim Jones
street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
city	Delray Beach	NIL	NIL
state	FL	AZ	MA
zip	33452	NIL	02134
zip4			
phonenumbers			
PhoneNumber [...]			
PhoneNumber[1]	NIL	NIL	NIL

For more information about editing the values, see "XML Editing" on page 237.

To set parameter element values:

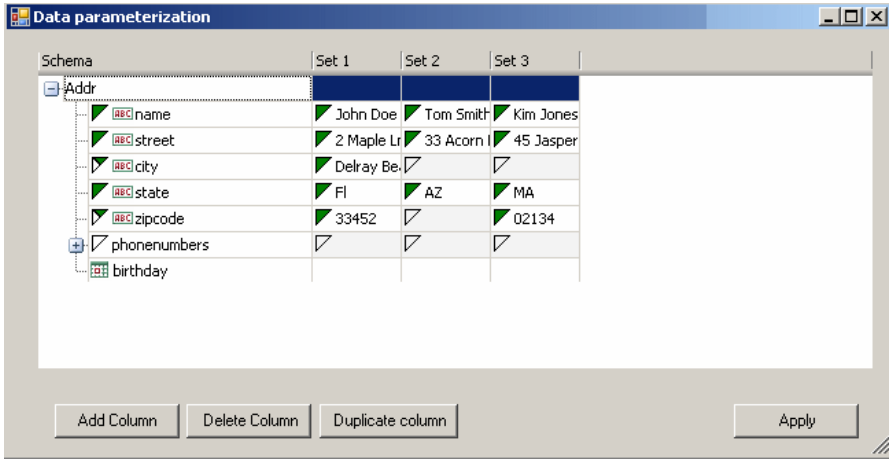
1 View the Parameter Properties.

If the Parameter Properties dialog box is not open, select **Vuser** > **Parameter List** and select the desired parameter. The dialog box shows a read-only view of the parameter values.



2 Open the Data Parameterization box.

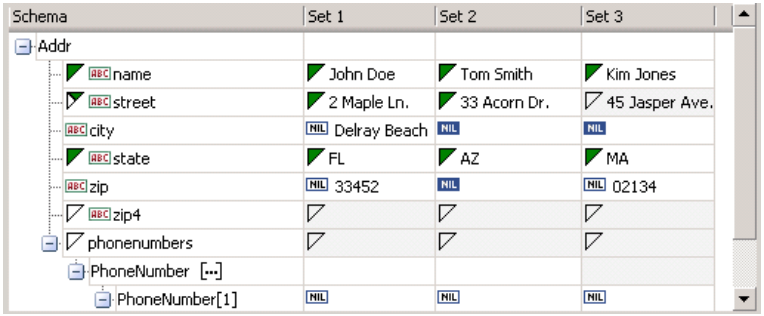
Click the **Edit Data** button to open the Data Parameterization dialog box.



3 Define value sets for the XML parameter.

In the **Set** columns, insert values corresponding to the schema.

If a row says **NIL**, it implies that the element is nillable. To include a value for the nillable element, enter the value as usual. To mark a value as **nil**, click the NIL icon to fill it in. This erases any value that you may have assigned to the element. In the following example, the **city** element is nillable, but it is only marked as nil in **Set 2** and **Set 3**—not in **Set 1**.

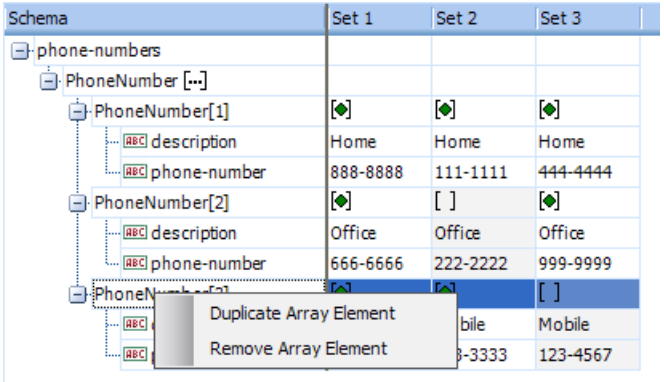


4 Create additional value sets.

To insert more value sets, click **Add Column** and insert another set of values in the new column. To copy an existing value set, select a row in the value set you want to copy and click **Duplicate Column**.

5 Copy arrays.

To duplicate an array element and its children, select the parent node and choose **Duplicate Array Element** from the right-click menu.

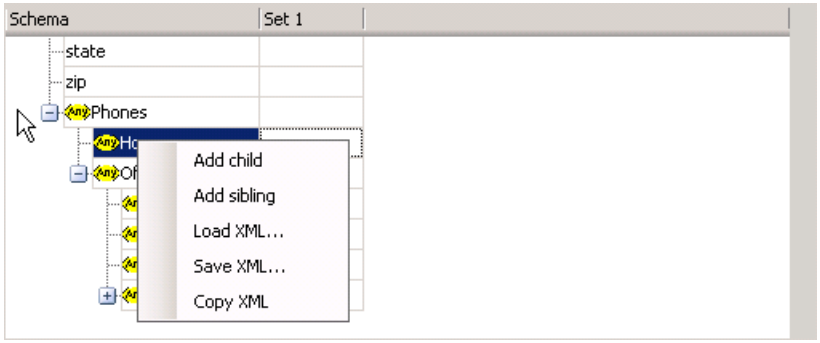


6 Handle the <any> elements.

For **any** type elements, right-click **<any>** in the **Schema** column and select one of the available options. These options may vary depending on the location of the cursor.

- **Add Array Element.** Adds a sub-element under the root element.
- **Insert child.** Adds a sub-element to the selected element.
- **Insert sibling.** Adds a sub-element on the same level as the selected element.
- **Load XML.** Loads the element values from an XML file.
- **Save XML.** Saves the array as an XML file.
- **Copy XML**. Copies the full XML of the selected element to the clipboard.

Click the **Rename** text to provide a meaningful name for each array element.



7 Remove unwanted columns.

To remove a value set, select it and click **Delete Column**.

8 Save the changes.

Click **Apply** to save the changes and update the view in the Parameter Properties dialog box.

Setting an Assignment Method

The assignment method indicates which of the value sets to use and how to use them. For example, you can instruct Vusers to use a new value set for each iteration and use the value sets sequentially or at random. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 1088.

To define an assignment method:

1 Open the Parameter Properties and select a parameter.

2 Define a data assignment method.

In the **Select next value** list, select a data assignment method to instruct the Vuser how to select the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Parameters" on page 1086.

3 Select an update option for the parameter.

In the **Update value on** list, select an update option. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 1088.

a 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. Select **Automatically allocate block size** or **Allocate** x **values for each Vuser.** For the second option, specify the number of values to allocate.

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

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

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

Modifying XML Parameter Properties

If you need modify a value set of a parameter, you can do so 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 in "Defining Value Sets" on page 1093.

72

Setting Parameter Properties

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

This chapter includes:

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 71, "File, Table, and XML Parameter Types."

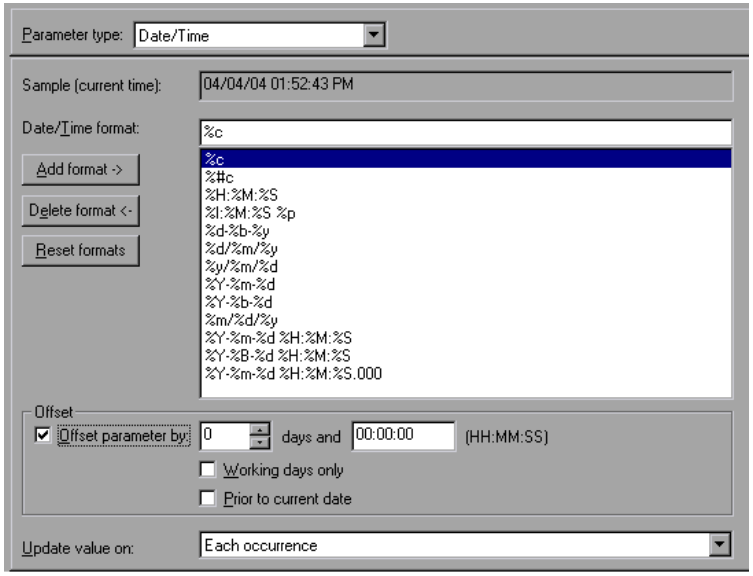
Setting Properties for Internal Data Parameter Types

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 1113.

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

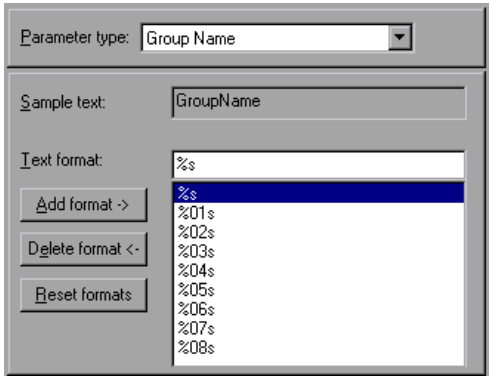
To instruct VuGen to use working day dates only, excluding weekends, select **Working days only**. To indicate a negative offset to test a date prior to the current, select **Prior to current date**.

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 1114.

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*.

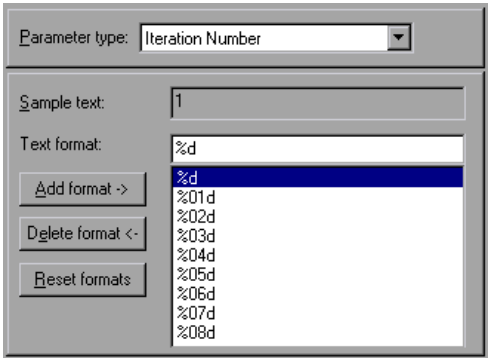


To set properties for the Group Name parameter type:

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 1113.
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.

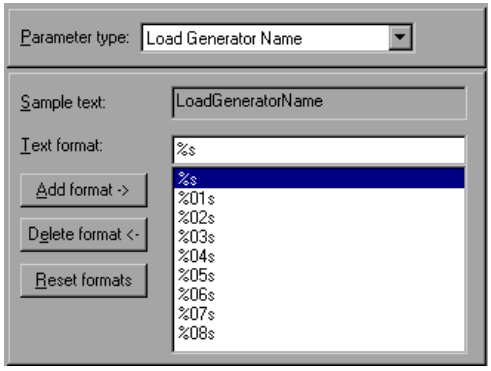


To set the properties for the Iteration Number parameter type:

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 1113.
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.



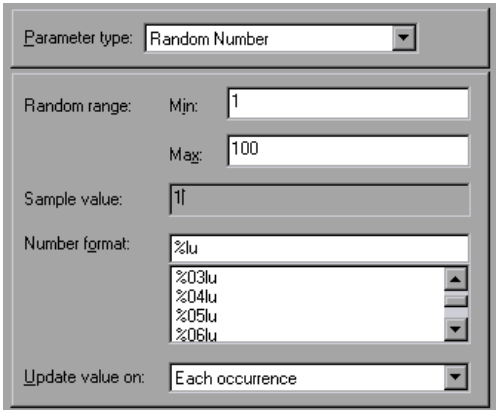
To set the properties for the Load Generator Name parameter type:

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 1113.
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:

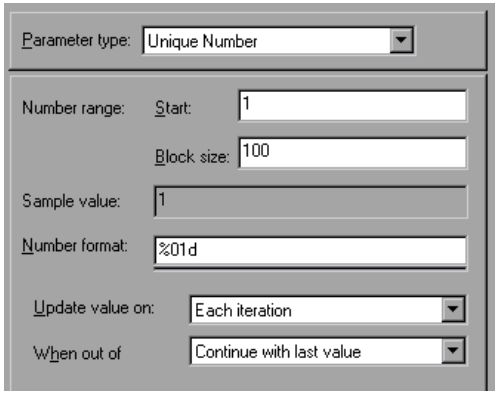
1. Enter a range defining the set of possible parameter values. You specify minimum and maximum values for the range of random numbers.
2. Select a **Number format**, indicating the length of the random number. Specify **%01lu** (or **%lu**) for one digit, **%02lu** for two digits, and so on. You can 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 1114.
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 1114.
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**.

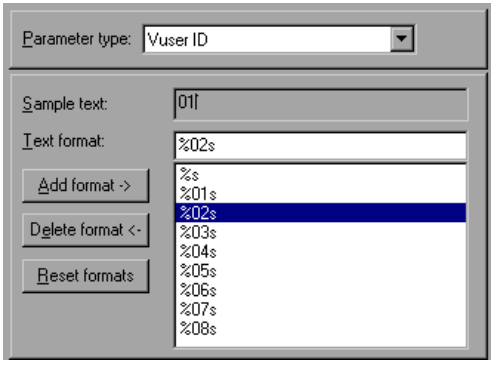
   Note: (For LoadRunner users!)
   When scheduling a scenario in the Controller, the **When out of values** option only applies to the **Run for HH:MM:SS** option in the Schedule Builder's Duration tab. It is ignored for the **Run until completion** option.

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.



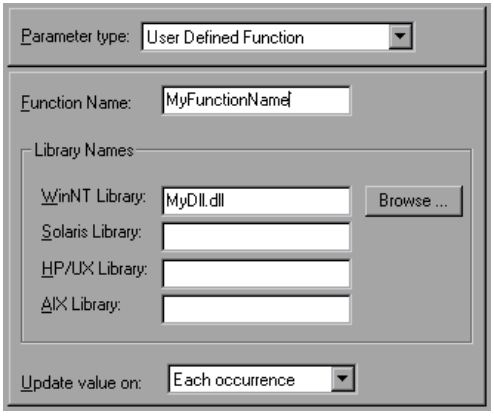
Note: This is not the ID number that appears in the Vuser window—it is a unique ID number generated at runtime.

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 1113.
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 1114.

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 1253.

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.

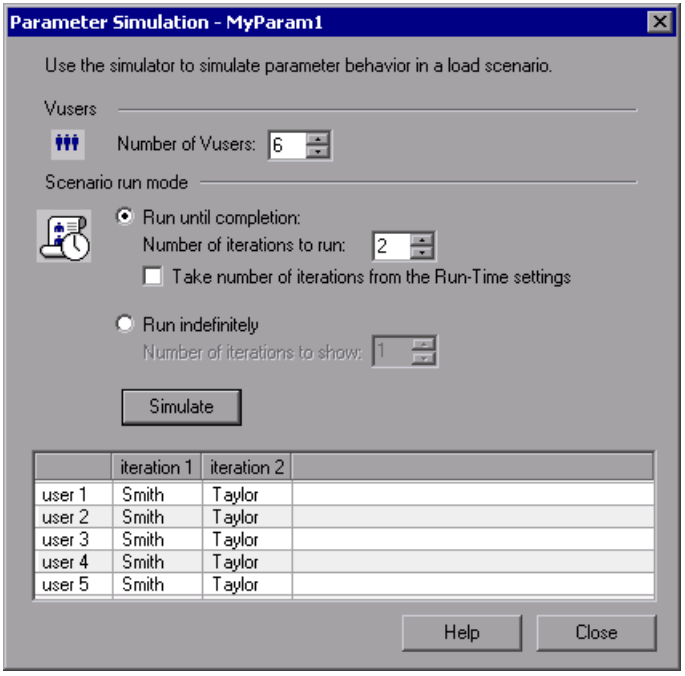
Simulating File Type Parameters

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 select **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 select **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 select **Run indefinitely**, you must define how many iterations to show. The number of iteration in the Run-Time settings is ignored. The number you select 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

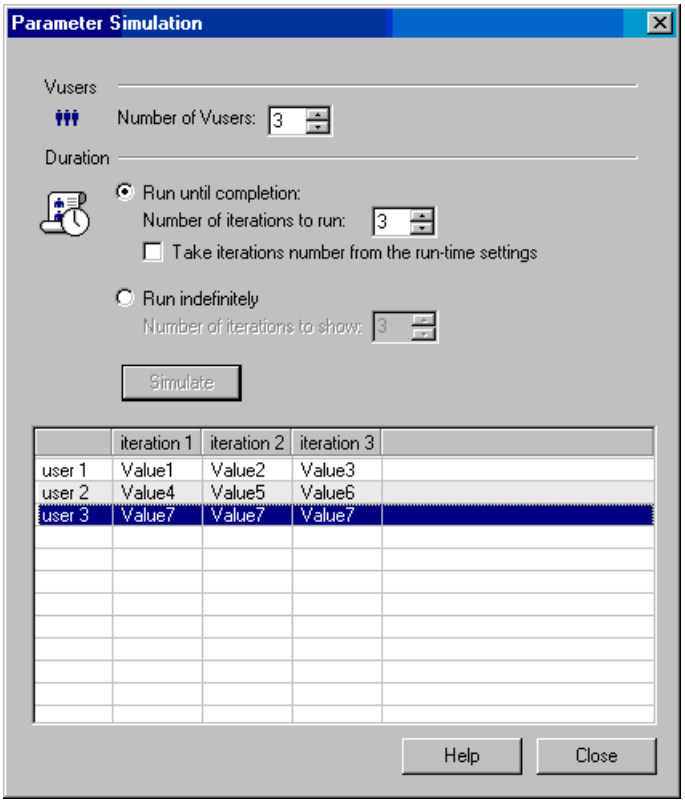
The following section illustrates how the File Parameter Simulator operates, demonstrating the difference between the Scenario run mode settings.

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.



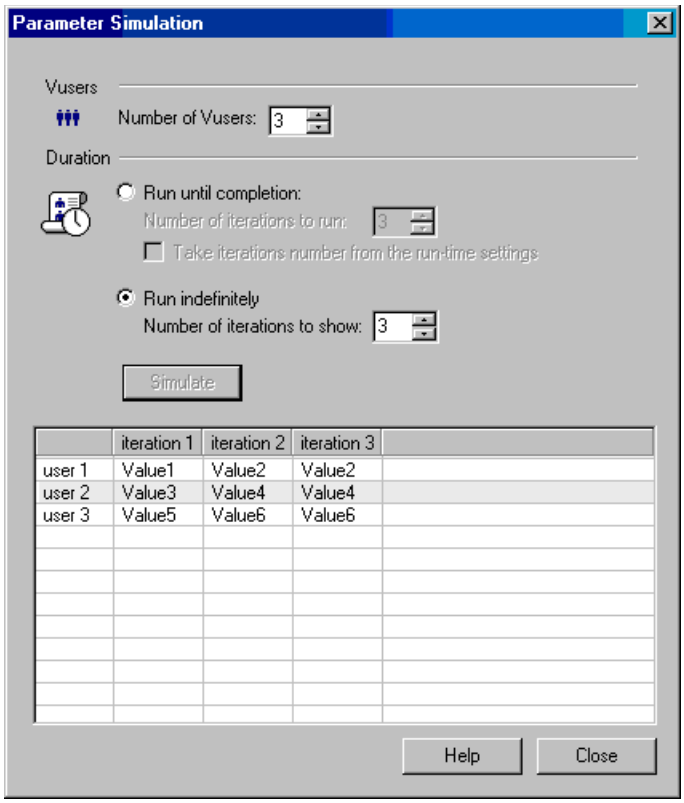
When the scenario run mode is set to **Run until completion**, the number of rows that each Vuser receives is the same as the number of iterations. The range allocation stops when there are no longer enough rows in the table.

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.

Part 4

Recording Options

73

Recording Options for Selected Protocols

This chapter contains information about recording options for selected protocols.

This chapter includes:

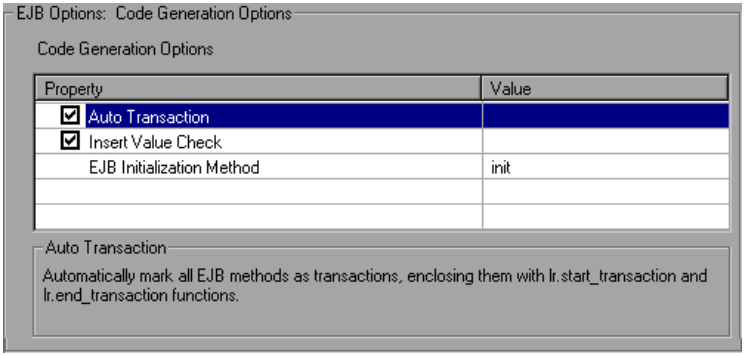
EJB Recording Options

You can set Classpath and Code Generation recording options for EJB Vusers. This section contains information about Code Generation recording options. For information about Classpath options, see Chapter 75, "Java Recording Options."

The **EJB Code Generation** options allow you to set properties in the area of automatic transactions and value checks. You can also indicate where to store the initialization method.

To set the EJB Code Generation recording options:

1 Click **Options** in the Start Recording dialog box. Select the **EJB Options:Code Generation Options** node in the Recording Options tree to edit the code generation options.



2 Enable the **Auto Transaction** option to automatically mark all EJB methods as transactions. This encloses all methods with **lr.start_transaction** and **lr.end_transaction** functions. By default, this option is enabled (true).

3 Enable the **Insert Value Check** option to automatically insert an **lr.value_check** function after each EJB method. This function checks for the expected return value for primitive values and strings.

4 Select an **EJB Initialization Method**. This is the method to which the EJB/JNDI initialization properties are written. The available methods are **init** (default) and **action**.

RDP Recording Options

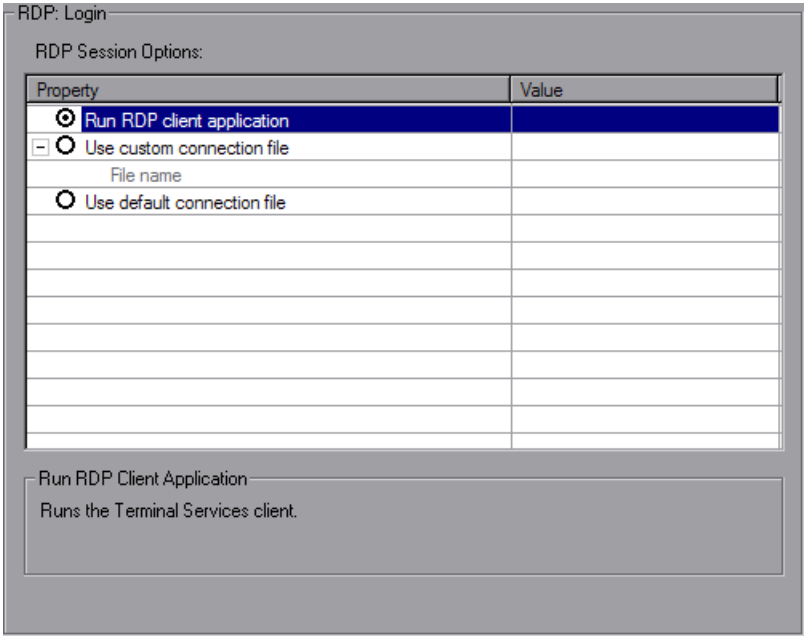
Before recording the script, you can configure how the RDP activities are recorded and how the script is generated. You do this by setting the Recording Options.

You can set the recording options in the following areas:

- "RDP Login Options" on page 1125
- "RDP Basic Code Generation Options" on page 1126
- "RDP Advanced Code Generation Options" on page 1128
- "RDP Agent Code Generation Options" on page 1129

RDP Login Options

In the RDP Login Options, you indicate how to run the Terminal Services client session.

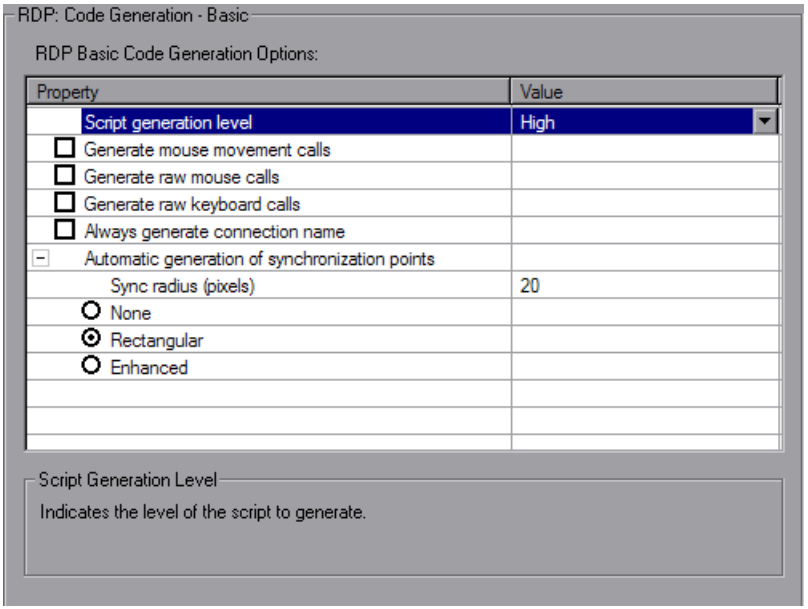


- **Run RDP client application.** Run the Terminal Services client.

- **Use custom connection file.** Run the Terminal Services client using an existing connection file. The file should have an ***.rdp** extension. You can browse for the file on your file system or network.
- **Use default connection file**. Use the **Default.rdp** file in your document's directory to connect.

RDP Basic Code Generation Options

The Basic Code Generation Options control the way VuGen creates a script—the level of detail, triggers, and timeouts.



You can set the following options:

- **Script generation level.** The level of the script and the type of API functions to use when generating the script. The available levels are **High**, **Low** or **Raw**.
  - **High.** Generate high level scripts. Keyboard events are translated to **rdp_type** calls. Two consecutive mouse clicks with the same coordinates are translated as a double-click.

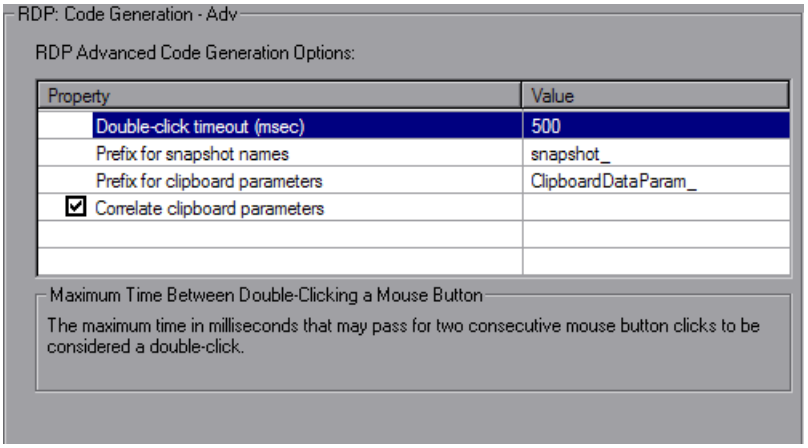
- **Low.** Generate low level scripts. Key up/down events are translated into rdp_key events. Modifier keys (Alt, Ctrl, Shift) are used as a KeyModifier parameter for other functions. Mouse up/down/ move events are translated to mouse click/drag events.
- **Raw.** Generates a script on a raw level, by extracting input events from network buffers and generating calls in their simplest form: key up/down, mouse up/down/move. The KeyModifier parameter is not used.

- **Generate mouse movement calls**. Generates **rdp_mouse_move** calls in the script. When enabled, this option significantly increases the script size (disabled by default).
- **Generate raw mouse calls**. If enabled, VuGen generates **rdp_mouse_button_up/down** calls as if the script level was set to Raw. Keyboard calls will still be generated according to the script level. If disabled, VuGen generates Mouse calls according to the script level. If the script level is set to Raw, this option is ignored (disabled by default).
- **Generate raw keyboard calls.** If enabled, VuGen generates **rdp_raw_key_up/down** calls as if the script level was set to Raw. Mouse calls will still be generated according to the script level. If disabled, VuGen generates Keyboard calls according to the script level. If the script level is set to Raw, this option is ignored (disabled by default).
- **Always generate connection name.** If selected, function call will contain the ConnectionName parameter. If not selected, the functions will only contain this parameter if more than a single **rdp_connect_server** appears in the script (disabled by default).
- **Automatic generation of syncronization points.** Synchronization points allow the script to pause in the replay while waiting for a window or dialog to pop-up or some other control to fulfil a certain condition. This option automatically generates **sync_on_image** functions before mouse clicks and drags (enabled by default). The **Sync radius** is the distance from the mouse operation to the sides of the rectangle which defines the synchronization area. The default is 20 pixels.
  - **None.** No synchronization points are automatically added.

- **Rectangular**. Creates synchronization points as rectangular boxes centered around the click or drag location. The **Sync Radius** is the distance from the mouse operation to the sides of the rectangle which defines the synchronization area. The default is 20 pixels.
- **Enhanced**. Creates synchronization points designed to select only the desired location (e.g. a button) and to react to changes in the UI (e.g. the button moves). If a synchronization region is not recognized, the rectangular synchronization settings are used.

RDP Advanced Code Generation Options

The Advanced Code Generation Options control the way VuGen creates a script - default parameter name prefixes and other options. Only advanced users are advised to modify these settings.



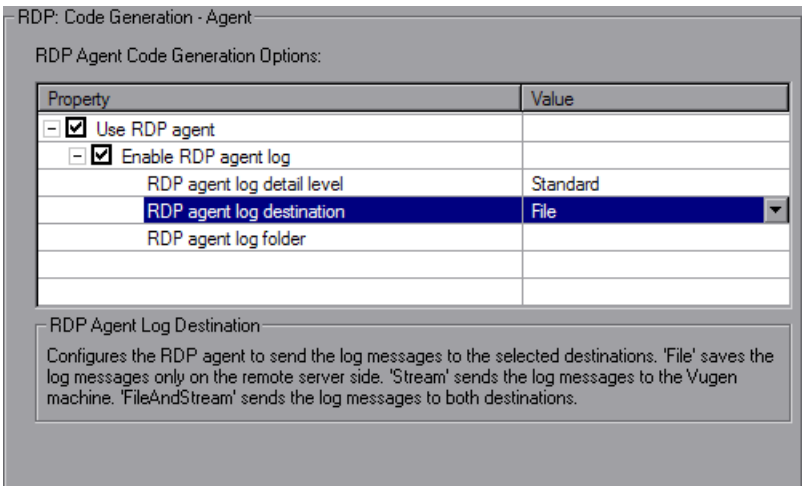
You can set the following options:

- **Double-click timeout (msec).** The maximum time in milliseconds between two consecutive mouse button clicks, to be considered a double-click. The default is 500 milliseconds.
- **Prefix for snapshot names.** The prefix for snapshot file names generated in the current script. This is useful when merging scripts—you can specify a different prefix for each script. The default is **snapshot_**.

- **Prefix for clipboard parameters.** The prefix for clipboard parameters generated in the current script. This is useful when merging scripts—you can specify a different prefix for each script. The default is **ClipboardDataParam_**.
- **Correlate clipboard parameters.** Replaces the recorded clipboard text sent by the user, with the correlated parameter containing the same text as received from the server.

RDP Agent Code Generation Options

The Agent Code Generation Options control the way the agent for Microsoft Agent for Terminal Server functions with VuGen during recording.



You can set the following options:

- **Use RDP agent.** Generates script using information gathered by the RDP agent during the recording session. LoadRunner RDP agent must be installed on the server.
- **Enable RDP agent log.** Enables the RDP agent log.
  - **RDP agent log detail level.** Configures the level of detail generated in the RDP agent log with **Standard** being the lowest level of detail and **Extended Debug** being the highest level of detail.

- **RDP agent log destination.** Configures the destination of the RDP agent log data. **File** saves the log messages only on the remote server side. **Stream** sends the log messages to the Vugen machine. **FileAndStream** sends the log messages to both destinations."

- **RDP agent log folder.** The folder path on the remote server that the RDP agent log file will be generated in.

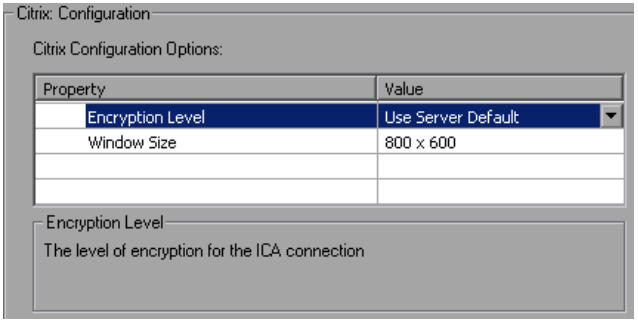
Understanding Citrix Recording Options

You can set the Citrix Recording options in the following areas.

- Configuration Recording Options
- Recorder Recording Options
- Code Generation Recording Options
- Login Recording Options (only for single protocol Citrix ICA scripts)

Configuration Recording Options

In the **Citrix:Configuration** Recording options, you set the window properties and encryption settings for the Citrix client during the recording session.

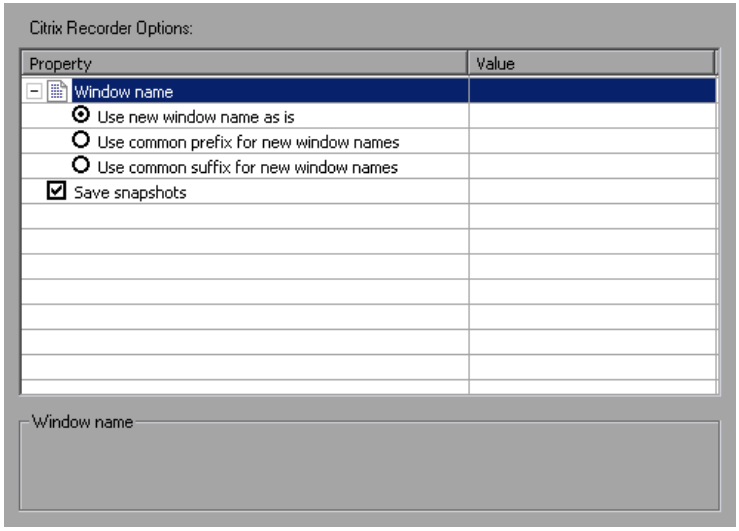


- **Encryption Level.** The level of encryption for the ICA connection: **Basic**, **128 bit for login only**, **40 bit**, **56 bit**, **128 bit**, or **Use Server Default** to use the machine's default.

➤ **Window Size.** The size of the client window: **640 x 480**, **800 x 600** (default), **1024 x 768, 1280 x 1024,** or **1600 x 1200**.

Recorder Recording Options

The **Citrix:Recorder** Recording options let you specify how to generate window names where the window titles change during recording. You can also specify whether to save snapshots of the screens together with the script files and whether to generate text synchronization functions.



Window Name

In some applications, the active window name changes while you are recording. If you try to replay the script as is, the Vuser uses the original window name and the replay may fail. Using the recording options, you can specify a naming convention for the windows in which VuGen uses a common prefix or common suffix to identify the window.

For example, if the original window's name is "untitled - Notepad" where the name changes during application's run to "my_test - Notepad", you can instruct VuGen to use the common suffix only, "Notepad".

The following options are available for generating window names during recording.

- **Use new window name as is.** Set the window name as it appears in the window title. (default)
- **Use common prefix for new window names.** Use the common string from the beginning of the window titles, as a window name.
- **Use common suffix for new window names.** Use the common string from the end of the window titles, as a name.

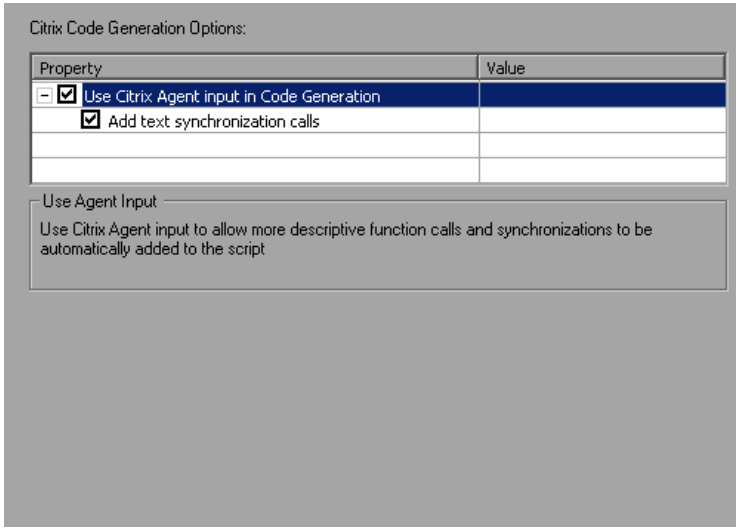
Note: Alternatively, you can modify the window names in the actual script after recording. In the Script view, locate the window name, and replace the beginning or end of the window name with the "*" wildcard notation.
ctrx_sync_on_window ("My Application*", ACTIVATE, …CTRX_LAST);

Snapshots

The **Save snapshots** option instructs VuGen to save a snapshot of the Citrix client window for each script step, when relevant. We recommend that you enable this option to provide you with a better understanding of the recorded actions. Saving snapshots, however, uses more disk space and slows down the recording session.

Code Generation Recording Options

The **Recording:Code Generation** Recording options let you configure the way VuGen captures information during recording.



- **Use Citrix Agent input in Code Generation.** Use the Citrix Agent input to generate a more descriptive script with additional synchronization functions (enabled by default).
- **Add text synchronization calls.** Adds text synchronization **Sync on Text** steps before each mouse click (disabled by default).

Text synchronization steps that you add manually during the recording are not affected by the above settings—they appear in the script even if you disable the above options. For more information about adding **Sync on Text** steps during recording, see "Manual Synchronization" on page 531.

The above options are also available for regenerating a script. For example, if you originally recorded a script with **Add text synchronization calls** disabled, you can regenerate after to recording to include text synchronization. For more information about regenerating your script, see "Regenerating a Vuser Script" on page 113.

Note: If you use the VuGen 8.1 or higher to regenerate a script from an earlier version of VuGen, the script will no longer be compatible with earlier versions—it behaves as if you created a new script.

Login Recording Options

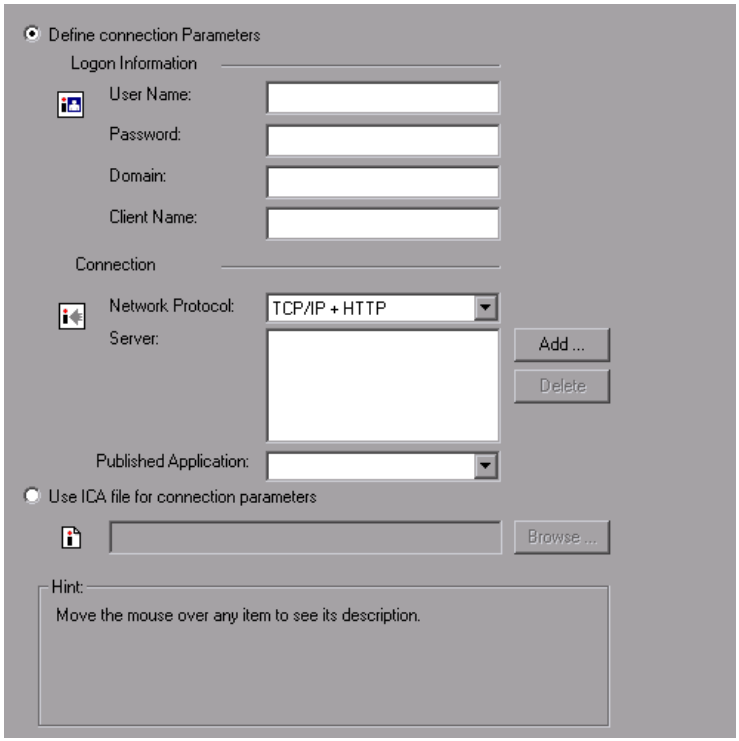
In the **Citrix:Login** Recording options, you set the connection and login information for the recording session. (When working with NFUSE, the Login options are not available since the login is done through the Web pages.)

You can provide direct login information or instruct VuGen to use an existing configuration stored in an **ica** file.

You must provide the name of the server—otherwise the connection VuGen generates a ctrx_connect_server function:

```
ctrx_connect_server("steel", "test", "test", "testlab", CTRX_LAST);
```

If you do not provide login information, you are prompted for the information when the client locates the specified server.



You provide the following user and server information for the Citrix session.

Logon Information. This section contains the login information:

- the **User Name** for the Citrix user.
- the **Password** for the Citrix user.
- the **Domain** of the Citrix user.
- the **Client Name**, by which the MetaFrame server identifies the client (optional).

Connection. This section contains the server information:

- **Network Protocol**. the preferred TCP/IP or TCP/IP+HTTP. Most Citrix Servers support TCP/IP. Certain servers, however, are configured by the administrators to allow only TCP/IP with specific HTTP headers. If you encounter a communication problem, select the TCP/IP+HTTP option.
- **Server.** The Citrix server name. To add a new server to the list, click **Add**, and enter the server name (and its port for TCP/IP + HTTP). Note that multiple servers apply only when you specify a Published Application. If you are connecting to the desktop without a specific application, then list only one server.
- **Published Application.** The name of the **Published Application** as it is recognized on Citrix server. The drop-down menu contains a list of the available applications. If you do not specify a published application, VuGen uses the server's desktop. If you added or renamed a published application, close the Recording options and reopen them to view the new list.

  To change the name of the published application on the Citrix client, you must make the change on the Citrix Server machine. Select **Manage Console** > **Application** and create a new application or rename an existing one.

  Note that if you do not specify a published application, Citrix load balancing will not work. To use load balancing when accessing the server's desktop, register the desktop as a published application on the server machine, and select this name from the **Published Application** drop-down list.

Using an ICA File with Connection Parameters

If you have an existing .ica file with all of the relevant configuration information, select **Use ICA file for connection parameters**. In the following row, specify the full path of the .ica file.

For information about the format of an ICA file, see "Understanding ICA Files" on page 536and the Citrix Website, www.citrix.com.

Setting the Citrix Recording Options

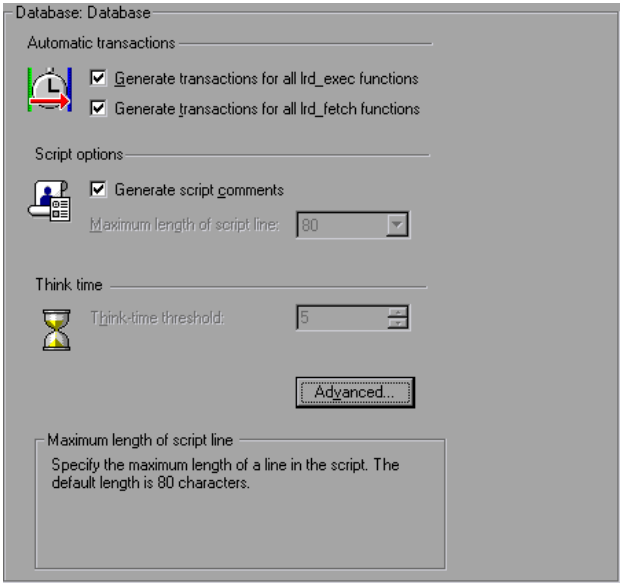
Before recording, you set the desired recording options.

To set the Citrix recording options:

1. Open the Recording Options dialog box. Select **Tools** > **Recording Options** or click the **Options** button in the Start Recording dialog box. The keyboard shortcut key is CTRL+F7.
2. Select the **Citrix:Login** node (only for single protocol Citrix ICA scripts).
   - ➤ If you have an existing ica file with all of the relevant configuration information, select **Use ICA file for connection parameters**. Specify the full path of the ica file, or click the **Browse** button and locate the file on the local disk or network.
   - ➤ If you do not have an ica file, select **Define connection parameters**. This is the default setting. Enter the **Connection** and **Identification** information.
3. Select the **Citrix:Configuration** node. Select an encryption level and a window size.
4. Select the **Citrix:Code Generation** node. To use information from the Citrix agent for a more descriptive script, enable the node's options.
5. Select the **Citrix:Recorder** node. Specify how to generate window names for windows whose titles change during the recording session.
6. To prevent VuGen from saving a snapshot for each step, clear **Save snapshots**.
7. When recording an NFUSE session, set the Web recording mode to URL-based. Select the **General:Recording** recording option and select **URL-based script**.
8. Click **OK** to accept the setting and close the dialog box.

Database Recording Options

Before you record a database session, you set the recording options. You can set basic recording options for automatic function generation, script options, and think time:



- **Automatic Transactions.** Marks every **lrd_exec** and **lrd_fetch** function as a transaction. When these options are enabled, VuGen inserts **lr_start_transaction** and **lr_end_transaction** around every **lrd_exec** or **lrd_fetch** function. By default, automatic transactions are disabled.
- **Script Options.** Generates comments into recorded scripts, describing the **lrd_stmt** option values. In addition, you can specify the maximum length of a line in the script. The default length is 80 characters.
- **Think Time.** VuGen automatically records the operator's think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an **lr_think_time** statement before LRD functions. If the recorded think time is below the threshold level, an **lr_think_time** statement is not generated. The default value is five seconds.

To set the Database recording options:

1. Select **Tools** > **Recording Options.** The Recording Options dialog box opens.
2. Select **Generate transactions for all lrd_exec functions** to enable automatic transactions for **lrd_exec** statements.

   Select **Generate transaction for all lrd_fetch functions** to enable automatic transactions for **lrd_fetch** statements.
3. Select **Generate script comments** to instruct VuGen to insert descriptive comments within the script.
4. To change the maximum length of a line in the VuGen editor, specify the desired value in the **Maximum length of script line** box.
5. To change the think-time threshold value from the five second default, specify the desired value in the **Think-time threshold** box.

You can also set advanced recording options relating to the trace level, Ctlib function generation, and the code generation buffer.

Database Advanced Recording Options

In addition to the basic recording options, you can configure advanced options for the log file detail, CtLib specific functions, buffer size, and the recording engine.

- **Recording Log Options.** You can set the detail level for the trace and ASCII log files. The available levels for the trace file are **Off**, **Error Trace**, **Brief Trace**, or **Full Trace**. The error trace only logs error messages. The Brief Trace logs errors and lists the functions generated during recording. The Full Trace logs all messages, notifications, and warnings.

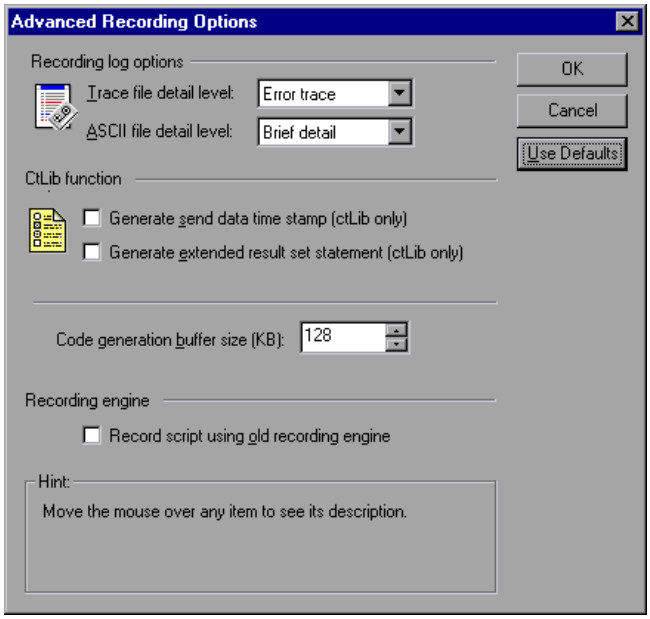
  You can also instruct VuGen to generate ASCII type logs of the recording session. The available levels are **Off**, **Brief detail**, and **Full detail**. The Brief detail logs all of the functions, and the Full detail logs all of the generated functions and messages in ASCII code.
- **CtLib Function Options.** You can instruct VuGen to generate a send data time stamp or an extended result set statement.

- **Time Stamp.** By default, VuGen generates **lrd_send_data** statements with the **TotalLen** and **Log** keywords for the **mpszReqSpec** parameter. The Advanced Recording Options dialog box lets you instruct VuGen to also generate the **TimeStamp** keyword. If you change this setting on an existing script, you must regenerate the Vuser script by choosing **Tools** > **Regenerate Script.** It is not recommended to generate the **Timestamp** keyword by default. The timestamp generated during recording is different than that generated during replay and script execution will fail. You should use this option only after a failed attempt in running a script, where an **lrd_result_set** following an **lrd_send_data** fails. The generated timestamp can now be correlated with a timestamp generated by an earlier **lrd_send_data**.
- **Extended Result Set.** By default, VuGen generates an **lrd_result_set** function when preparing the result set. This setting instructs VuGen to generate the extended form of the **lrd_result_set** function, **lrd_result_set_ext**. In addition to preparing a result set, this function also issues a return code and type from **ct_results**.

- **Code Generation Buffer Size.** Specify in kilobytes the maximum size of the code generation buffer. The default value is 128 kilobytes. For long database sessions, you can specify a larger size.
- **Recording Engine.** You can instruct VuGen to record scripts with the older LRD recording engine for compatibility with previous versions of VuGen. This option is only available for single-protocol scripts.

To set advanced recording options:

1 Click the **Advanced** button in the Database node of the Recording Options dialog box. The Advanced Recording Options dialog box opens.



2 Select a **Trace file detail level**. To disable the trace file, select **Off**.

3 To generate an ASCII log file, select the desired detail level from the **ASCII file detail level** box.

4 For CtLib: To instruct VuGen to generate the TimeStamp keyword for **lrd_send_data** functions, select the **Generate send data time stamp** option.

5 For CtLib: To instruct VuGen to generate **lrd_result_set_ext** instead of **lrd_result_set**, select the **Generate extended result set statement** option.

6 To modify the size of the code generation buffer from the default value of 128 kilobytes, enter the desired value in the **Code generation buffer size** box.

7 To use the old VuGen recording engine to allow backwards compatibility, select the **Record script using old recording engine** option.

8 Click **OK** to save your settings and close the Advanced Recording Options dialog box.

Setting the AMF Recording Mode

You can instruct VuGen how to generate a script from a Flash Remoting session using the AMF and Web Protocols. The options are:

- AMF and Web
- AMF Only
- Web Only

By default, VuGen generates only AMF calls in the script. To configure the recorded protocols, open the Recording options (**Tools > Recording Options**) and select the **General:Protocols** node. For more information, see "Adding and Removing Protocols" on page 97.

Note: If you record with one of the above options, you can regenerate the script afterwards to include or exclude other protocols. For more information about regenerating your script, see "Regenerating a Vuser Script" on page 113.

AMF and Web

If you enable both AMF and Web protocols, VuGen generates functions for the entire business process. When it encounters AMF data, it generates the appropriate AMF functions.

In the following segment, VuGen generated both Web (**web_url**) and AMF (**amf_call**, **amf_define_envelope_header_set**) functions.

```
web_url("flash",
    "URL=http://testlab:8200/flash/", "Resource=0",
    …
    "Snapshot=t1.inf",
    EXTRARES,
    "Url=movies/XMLExample.swf", "Referer=", ENDITEM,
    "Url=movies/JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);

web_link("Sample JavaBean Movie Source",
    "Text=Sample JavaBean Movie Source",
    "Snapshot=t2.inf",
    EXTRARES,
    "Url=XMLExample.swf", "Referer=", ENDITEM,
    "Url=JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);

amf_set_version("0");

amf_define_header_set("Id=amf_header_set",
    HEADER,
    "Name=amf_server_debug",
    "MustUnderstand=true",
    "Data=<object><boolean key=\"coldfusion\">true</boolean>
            <boolean key=\"""amfheaders\">false</boolean>…
    LAST);

amf_call("flashgateway.samples.FlashJavaBean.testDocument",
    "Gateway=http://testlab:8200/flashservices/gateway",
    "AMFHeaderSetId=amf_header_set",
    "Snapshot=t3.inf",
    MESSAGE,
    "Method=flashgateway.samples.FlashJavaBean.testDocument",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    "<xmlString><![CDATA[<TEST message=\"test\"><INSIDETEST/>
      </TEST>]]></""xmlString>",
    END_ARGUMENTS,
    LAST);
```

AMF Only

If you are just interested in the AMF calls to emulate the Flash Remoting, you can disable the Web calls and only generate the AMF calls.

The following example shows the above session recorded with the AMF protocol enabled and the Web protocol disabled.

```
Action()
{
amf_set_version("0");

amf_define_header_set("Id=amf_header_set",
    HEADER,
    "Name=amf_server_debug",
    "MustUnderstand=true",
    "Data=<object><boolean key=\"coldfusion\">true</boolean>
            <boolean key=\"""amfheaders\">false</boolean>…
    LAST);

amf_call("flashgateway.samples.FlashJavaBean.testDocument",
    "Gateway=http://testlab:8200/flashservices/gateway",
    "AMFHeaderSetId=amf_header_set",
    "Snapshot=t3.inf",
    MESSAGE,
    "Method=flashgateway.samples.FlashJavaBean.testDocument",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    "<xmlString><![CDATA[<TEST message=\"test\"><INSIDETEST
      /></TEST>]]></"
    "xmlString>",
    END_ARGUMENTS,
    LAST);
…
```

Note that this recording method may not represent a complete business process—it only displays the Flash Remoting calls that use AMF.

Web Only

The Web Only option provides a fallback to the Web HTTP technology—VuGen does not generate any AMF calls. Instead it generates **web_custom_request** functions with the Flash Remoting information.

The following shows the above segment regenerated without AMF:

```
web_url("flash",
    "URL=http://testlab:8200/flash/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    EXTRARES,
    "Url=movies/XMLExample.swf", "Referer=", ENDITEM,
    "Url=movies/JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);

web_link("Sample JavaBean Movie Source",
    "Text=Sample JavaBean Movie Source",
    "Snapshot=t2.inf",
    EXTRARES,
    "Url=XMLExample.swf", "Referer=", ENDITEM,
    "Url=JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);

web_custom_request("gateway",
    "URL=http://testlab:8200/flashservices/gateway",
    "Method=POST",
    "Resource=0",
    "RecContentType=application/x-amf",
    "Referer=",
    "Snapshot=t3.inf",
    "Mode=HTML",
    "EncType=application/x-amf",

"BodyBinary=\\x00\\x00\\x00\\x01\\x00\\x10amf_server_debug\\x01\\x00\\x00\\x00`\\x0
3\\x00\ncoldfusion\\x01\\x01\\x00\namfheaders\\x01\\x00\\x00\\x03amf\\x01\\x00\\x00\\
x0Bhttpheaders\\x01\\x00\\x00\trecordset\\x01\\x01\\x00\\x05error\\x01\\x01\\x00\\x05tr
ace\\x01\\x01\\x00\\x07m_debug\\x01\\x01\\x00\\x00\t\\x00\\x01\\x00/flashgateway.sam
ples.FlashJavaBean.testDocument\\x00\\x02/1\\x00\\x00\\x004\n\\x00\\x00\\x00\\x01\\x
0F\\x00\\x00\\x00*<TEST message=\"test\"><INSIDETEST /></TEST>",
        LAST);
```

AMF Code Generation Options

When recording Flex applications, in certain cases VuGen may be unable to decode externalizable objects. This is due to a proprietary encoding scheme employed by the Flex 2 application.

To overcome this issue, you can instruct VuGen to generate a Custom Request function in the case the code is not decipherable.

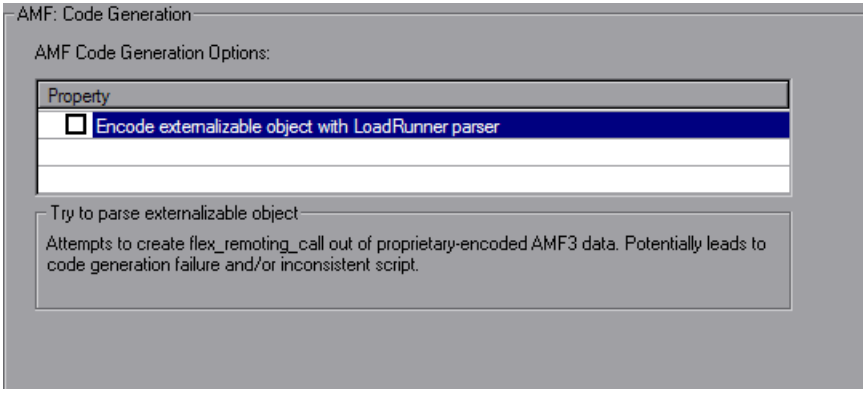
The AMF Recording option instructs VuGen to attempt to generate AMF Call requests for externalizable objects. It decodes all externalizable objects as standard AMF3 objects, generating **amf_call** functions.

If you disable this option (default), VuGen generates a Custom Request function containing unparsed AMF3 binary data when encountering an externalizable object.

Note: Enabling this option may reduce the stability of code generation.

To set the AMF code generation for externalizable objects:

1. Open the Recording Options dialog box. Select **Tools > Recording Options** or click the **Options** button in the Start Recording dialog box. The keyboard shortcut key is CTRL+F7.
2. Select the **AMF:Code Generation** node. To generate standard **amf_call** functions for externalizable objects, enable the recording option.



Flex Code Generation Options

When recording Flex applications, in certain cases, VuGen may be unable to decode externalizable objects. This is due to a proprietary encoding scheme employed by the Flex 2 application.

To overcome this issue, you can instruct VuGen to generate a **flex_web_request** function in case the code is not decipherable.

There are two options to overcome this issue:

1) You can use the LoadRunner parser to attempt to generate Flex Remote requests for externalizable objects. It decodes all externalizable objects as standard AMF3 objects, generating flex functions.

2) You can use your server's parser to attempt to generate Flex Remote requests for externalizable objects. It decodes all externalizable objects as standard AMF3 objects, generating **flex_amf_call** functions.

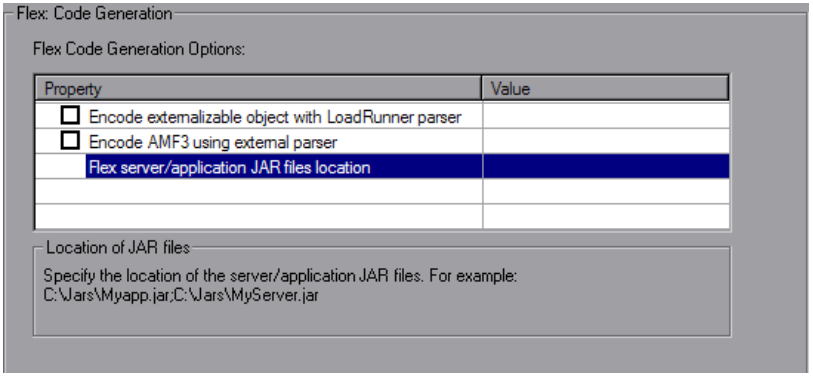
If you disable both options (default), VuGen generates a **flex_web_request** function containing unparsed AMF3 binary data when encountering an externalizable object.

Note: Enabling the encoding using the LoadRunner parse may reduce the stability of code generation.

To set the Flex code generation for externalizable objects:

1 Open the Recording Options dialog box. Select **Tools** > **Recording Options** or click the **Options** button in the Start Recording dialog box.

2 Select the **Flex:Code Generation** node. Check **Encode externalizable object with LoadRunner parser** and click **OK**.



3 Regenerate your script. Check for errors in the code gereration log. If there were no **web_custom_request** or no **flex_custom_request** steps in the script, the issue has been solved and procedure is complete.

4 If the above step was not successful, go back to **Tools** > **Recording Options** and select the **Flex:Code Generation** node.

5 Check **Encode AMF3 using external parser**.

6 Copy the server and application jars to any directory on your machine and to the same directory on the load generator. For information about which servers are supported and which jars to use contact HP software support.

7 Select **Tools** > **Recording Options** and select the **Flex:Code Generation** node. To the right of the **Flex server/application JAR files location** option, specify the location of each jar file on your machine seperated with semicolons. For example C:\Jars\Myapp.jar;C:\Jars\MyServer.jar.

8 Click **OK**.

9 Regenerate your script. Check for errors in the code gereration log. If there were no **web_custom_request** or no **flex_custom_request** steps in the script, the issue has been solved and procedure is complete. If not, contact HP software support.

Microsoft .NET Recording Options

You can set both Script and Microsoft .NET-specific recording options. This section describes the recording options specific to Microsoft .NET. For information on the Script recording options, see "Setting Script Generation Preferences" on page 1211

The recording options specific to Microsoft .NET are in the area of recording settings and .NET filters.

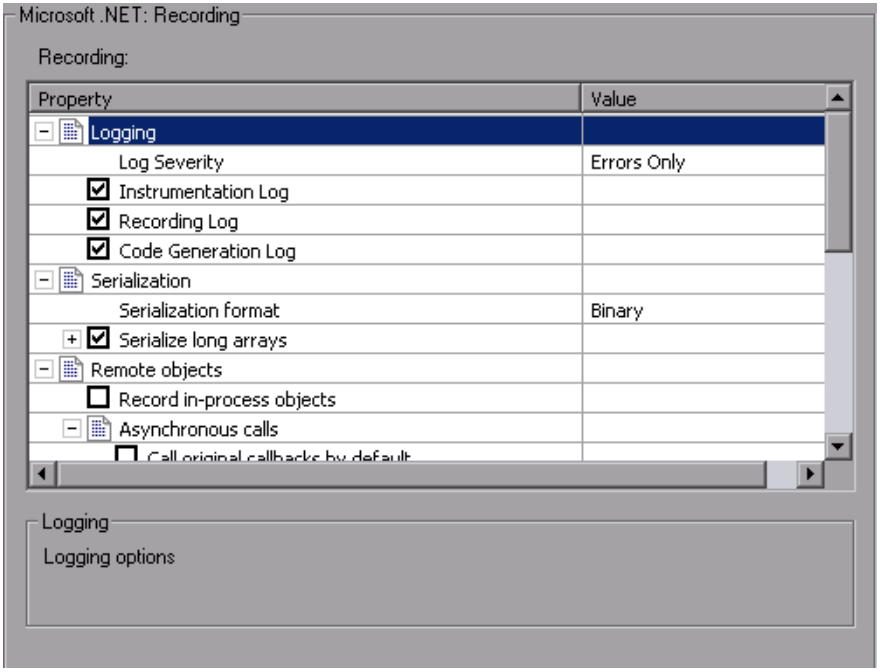
The recording settings let you control the logging, serialization, debugging, and the trace level of the logging. For more information, see "To open the .NET Recording Options dialog box, select Tools > Recording Options and select the Recording node." on page 1150.

The filtering options let you select a filter for the recording, using a standard .NET Remoting, ADO.NET, Enterprise Services, or WCF filter. You can also create custom filters and configure them according to your needs. For more information, see Chapter 47, "Microsoft .NET Filters".

For information about recording WCF duplex communication, see "WCF Recording Options" on page 1155.

Some of the recording options are also relevant to code regeneration. After recording a script with certain options, you can modify the code generation options and regenerate the script with different settings or even in a different language. To change the options and regenerate a script, select **Tools** > **Regenerate Script**. In the Regenerate Script dialog box, click **Options**. For more information, see "Regenerating a Vuser Script" on page 113.

To open the .NET Recording Options dialog box, select **Tools** > **Recording Options** and select the **Recording** node.



You can configure your script in the following areas:

- Logging
- Serialization Settings
- Remote Objects
- WCF Duplex Binding
- Debug Options

- Filters
- Code Generation

Logging

- The Logging options let you set the level of detail in the recording log file: **Log Severity**. Sets the level of logging to **Errors Only** (default), or **Debug**. The severity setting applies for all the logs that you enable below. You should always use the **Errors Only** log unless specifically instructed to do otherwise by HP support, since detailed logging may significantly increase the recording time.
- **Instrumentation Log.** Logs messages related to the instrumentation process (enabled by default).
- **Recording Log.** Logs messages issued during recording (enabled by default).
- **Code Generation Log.** Logs messages issued during the code generation stage (enabled by default).

Serialization Settings

The Serialization settings let you set the serialization format.

VuGen uses serialization when it encounters an unknown object during the recording, provided that the object supports serialization. An unknown object can be an input argument which was not included by the filter and therefore its construction was not recorded. Serialization helps prevent compilation errors caused by the passing of an unknown argument to a method. If an object is serialized, it is often advisable to set a custom filter to record this object.

- **Serialization format.** The format of the serialization file that VuGen creates while recording a class that supports serialization: **Binary**, **XML**, or **Both**. The advantage of the binary format is that since it is more compressed, it is quicker. The disadvantage of the binary format is that you do not have the ability to manipulate the data as you do with XML.

- **Serialize long arrays.** For long arrays containing serializable objects (for example, an array of primitives), use VuGen's serialization mechanism. Enabling this option generates LrReplayUtils.GetSerializedObject calls if the array size is equal to or larger than the threshold value.
- **Threshold value for long array size.** The threshold size for an array to be considered a **long** array. If the array size is equal to or larger than this size, VuGen serializes it when detecting serializable objects.

Tip: For XML serialization, you can view the content of the XML file. To view the file, select **View XML** from the right-click menu.

Remote Objects

This section lets you define the proxy-related recording options.

- **Record in-process objects.** Records activity between the client and server when the server is hosted in the same process as the client. Since the actions are not true client/server traffic, it is usually not of interest. When in-process methods are relevant, for example, in certain Enterprise Service applications, you can enable this option to capture them (disabled by default).

Asynchronous Calls

In the following section, you specify how VuGen should handle asynchronous calls on remote objects and their callback methods. These options are mostly relevant for .NET Remoting and WCF environments.

- **Call original callbacks by default.** Uses the recorded application's original callback when generating and replaying the script. If the callback method is explicitly excluded by a filter, the callback will be excluded even if you enable this option. If your callbacks perform actions that are not directly related to the business process, such as updating the GUI, then make sure to disable this option.
- **Generate asynchronous callbacks.** This option defines how VuGen will handle callbacks when the original callbacks are not recorded. This is relevant when the above option, **Call original callbacks** is disabled or when the callbacks are explicitly excluded.

When you enable this option, it creates a dummy method which will be called during replay instead of the original callback. This dummy callback will be generated in the **callbacks.cs** section of the script.

When you disable this option, VuGen inserts a NULL value for the callback and records the events as they occur.

The following segment shows script generation for a Calculator client, when **Generate asynchronous callbacks** is enabled.

```
lr.log("Event 2: CalculatorClient_1.Add(2, 3);");
Int32RetVal = CalculatorClient_1.Add(2, 3);
// Int32RetVal = 5;

callback_1 = new AsyncCallback(this.OnComplete1);
lr.log("Event 3: CalculatorClient_1.BeginAdd(2, 3, callback_1, null);");
IAsyncResult_1 = CalculatorClient_1.BeginAdd(2, 3, callback_1, null);
```

To display the callback method, OnComplete1, you click on the **callback.cs** file in the left pane.

The following segment shows script generation when the option is disabled. VuGen generates a NULL in place of the callback and records the events of the callback as they occur.

```
lr.log("Event 3: CalculatorClient_1.BeginAdd(2, 3, null, null);");
IAsyncResult_1 = CalculatorClient_1.BeginAdd(2, 3, null, null);

lr.log("Event 5: CalculatorClient_1.EndAdd(IAsyncResult_1);");
Int32RetVal = CalculatorClient_1.EndAdd(IAsyncResult_1);
// Int32RetVal = 5;

lr.log("Event 6: ((ManualResetEvent)(IAsyncResult_1.AsyncWaitHandle));");
ManualResetEvent_1 = ((ManualResetEvent)(IAsyncResult_1.AsyncWaitHandle));

lr.log("Event 7: ManualResetEvent_1.Close();");
ManualResetEvent_1.Close();
```

Note: If you recorded a script with specific recording options, and you want to modify them, you do not need to re-record the script. Instead you can regenerate the script with the new settings. For more information, see "Regenerating a Vuser Script" on page 113.

WCF Duplex Binding

You can set options for applications that use the WCF (Windows Communication Foundation). For more information, see "Recording WCF Duplex Communication" on page 743.

Debug Options

The debug options allow you to trace the stack and specify its size.

- **Stack Trace.** Traces the contents of the stack for each invocation within the script. It allows you to determine which classes and methods were used by your application. This can be useful in determining which references, namespaces, classes, or methods to include in your filter. Enabling the trace may affect your application's performance during recording. This trace is disabled by default.
- **Stack Trace Limit.** The maximum number of calls to be stored in the stack. The default is 20 calls. If the number of calls exceeds the limit, VuGen truncates it.

Filters

Ignore all assemblies by default. Ignore all assemblies that are not explicitly included by the selected filter. If you disable this option, VuGen looks for a matching filter rule for all assemblies loaded during the recording.

Code Generation

The Code Generation options let you indicate whether to show warnings and a stack trace during code generation.

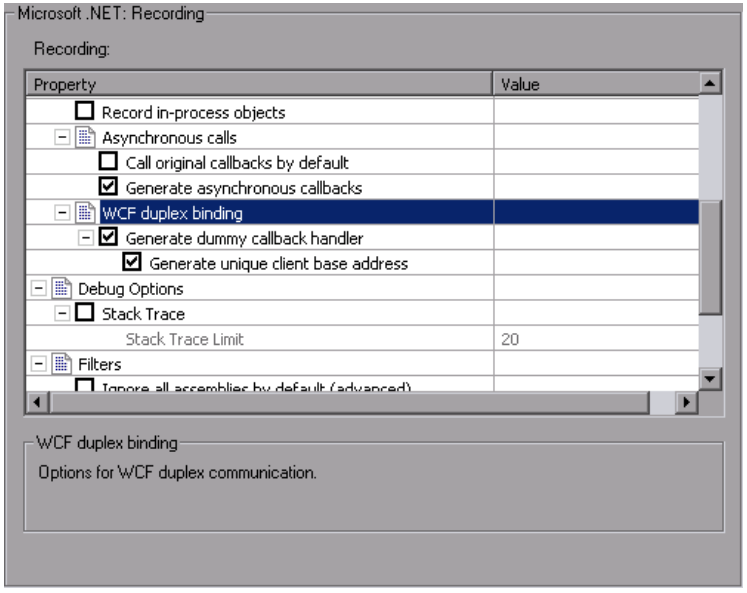
- **Show Warnings.** Shows warning messages that are issued during the code generation process.

- **Show Stack Trace.** Shows the Recorded stack trace if it is available.
- **Show All Event Subscriptions.** Generate code for all event subscriptions that were recorded (disabled by default). If this option is disabled, VuGen will only generate code for events in which both the publisher (the object which invokes the event) and the subscriber (the object informed of the event) are included in the filter.

WCF Recording Options

VuGen's recording options for WCF's duplex communications enable you to generate a script that will be effective for load testing. You can set recording options on the following areas:

- Generating Dummy Callback Implementations
- Recording Dual HTTP Bindings



Generating Dummy Callback Implementations

The **Generate Dummy Callback Handler** recording option instructs VuGen to replace the original callback in duplex communication with a dummy callback.

The dummy callback implementation performs the following actions:

- **Store arguments.** When the server calls the handler during replay, it saves the method arguments to a key-value in memory map.
- **Synchronize replay.** It stops the script execution until the next response arrives. VuGen places the synchronization at the point that the callback occurred during recording. This is represented in the script by a warning:

```
#warning:  Code Generation Warning
    // Wait here for the next response.
    // The original callback during record was:…
```

As part of the synchronization, the script calls GetNextResponse to get the stored value.

```
Vuser<Callback_Name>.GetNextResponse();
```

Enabling the Dummy Callback Recording Option

By default, this option is enabled.

To enable this recording option:

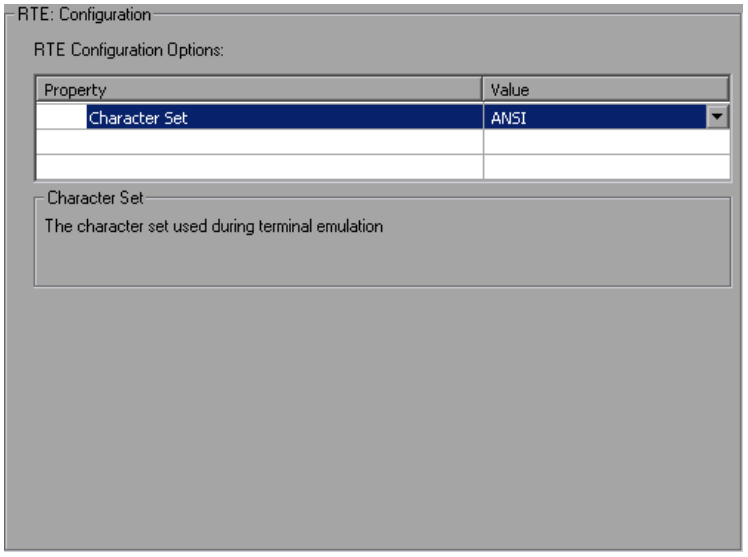
1. Select **Tools** > **Recording Options**.
2. Select the **Microsoft .NET:Recording** node.

Select **Generate Dummy Callback Handler**.

RTE Configuration Options

You can set the recording options to match the character set used during terminal emulation. The default character set is ANSI. For Kanji and other multi-byte platforms, you can specify DBCS (Double-byte Character Set).

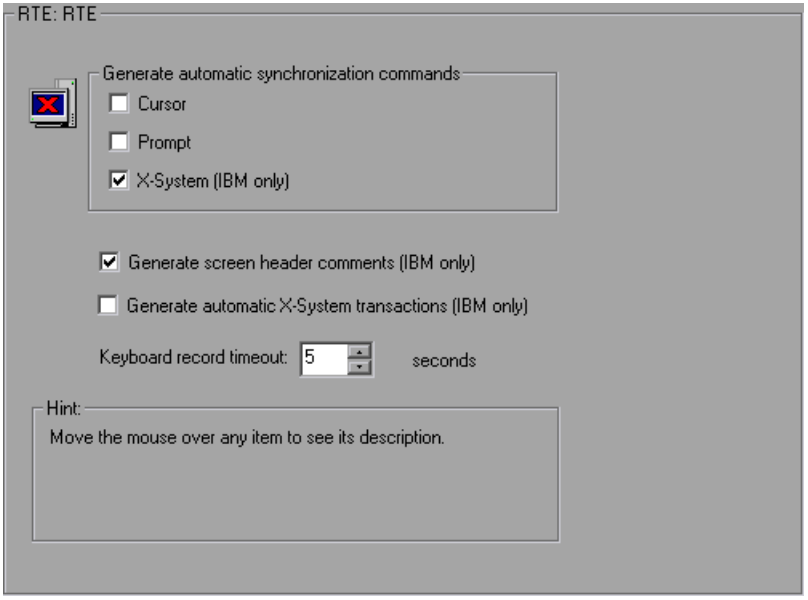
To open the Configuration Recording Options, click the **Recording Options** button on the toolbar or select **Tools** > **Recording Options**. Select the **RTE:Configuration** node.



RTE Recording Options

By setting the recording options, you can customize the code that VuGen generates for RTE functions. You use the Recording Options dialog box to set the recording options.

To open the Recording Options dialog box, click the **Recording Options** button on the toolbar, or select **Tools** > **Recording Options**. Select the **RTE:RTE** node.



You can set the following recording options:

- Automatic Synchronization Commands
- Automatic Screen Header Comments (IBM terminals only)
- Automatic X-System Transactions (IBM terminals only)
- Keyboard Recording Timeout

Automatic Synchronization Commands

VuGen can automatically generate a number of TE-synchronization functions, and insert them into the script while you record.

1 You can specify that VuGen generate a **TE_wait_sync** function each time a new screen is displayed while recording. To do so, select the **X-System** check box in the Recording Options dialog box.

By default, VuGen does automatically generate a **TE_wait_sync** function each time a new screen is displayed while recording.

Note: VuGen generates **TE_wait_sync** functions when recording IBM block mode terminals only.

2 You can specify that VuGen generate a **TE_wait_cursor** function before each **TE_type** function. To do so, select the **Cursor** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate **TE_wait_cursor** functions.

3 You can specify that VuGen generate a **TE_wait_text** function before each **TE_type** function (where appropriate). To do so, select the **Prompt** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate a **TE_wait_text** function before each **TE_type** function.

Note: VuGen generates meaningful **TE_wait_text** functions when recording VT type terminals only. Do not use automatic **TE_wait_text** function generation when recording block-mode (IBM) terminals.

Automatic Screen Header Comments (IBM terminals only)

You can instruct VuGen to automatically generate screen header comments while recording a Vuser script, and insert the comments into the script.

Generated comments make a recorded script easier to read by identifying each new screen as it is displayed in the terminal emulator. A generated comment contains the text that appears on the first line of the terminal emulator window. The following generated comment shows that the Office Tasks screen was displayed in the terminal emulator:

```
/* OFCTSK                Office Tasks                */
```

To make sure that VuGen automatically generates comments while you record a script, select **Generate screen header comments** in the Recording Options dialog box.

By default, VuGen does not automatically generate screen comments.

Note: You can generate comments automatically only when using block-mode terminal emulators such as the IBM 5250.

Automatic X-System Transactions (IBM terminals only)

You can specify that VuGen record the time that the system was in the X SYSTEM mode during a scenario run. To do so, VuGen inserts a **TE_wait_sync_transaction** function after each **TE_wait_sync** function. Each **TE_wait_sync_transaction** function creates a transaction with the name "default". Each **TE_wait_sync_transaction** function records the time that the system spent in the previous X SYSTEM state.

To instruct VuGen to insert **TE_wait_sync_transaction** statements while recording, select the **Generate automatic X SYSTEM transactions** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate transactions.

Keyboard Recording Timeout

When you type text into a terminal emulator while recording, VuGen monitors the text input. After each keystroke, VuGen waits up to a specified amount of time for the next key stroke. If there is no subsequent keystroke within the specified time, VuGen assumes that the command is complete. VuGen then generates and inserts the appropriate **TE_type** function into the script.

To set the maximum amount of time that VuGen waits between successive keystrokes, enter an amount in the **Keyboard record timeout** box.

By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function.

SAPGUI Recording Options

You use the recording options to set your SAP-related preferences for the recording session. To open the Recording Options dialog box, select **Tools** > **Recording Options** or click **Options** in the Start Recording dialog box. The keyboard shortcut is CTRL+F7.

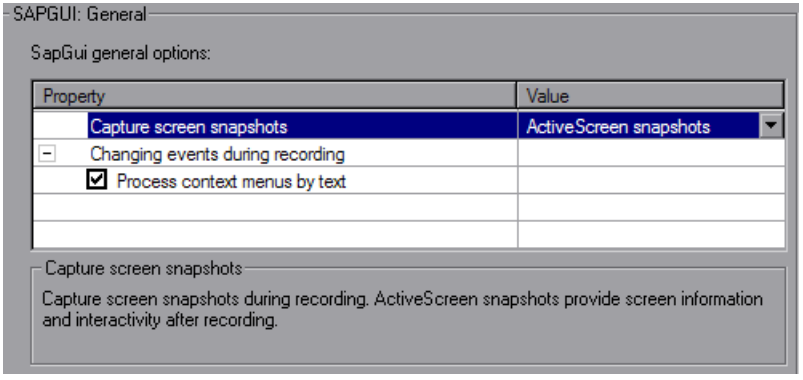
You can set recording options in the following areas:

- SAPGUI General Recording Options
- SAPGUI Code Generation Recording Options
- SAPGUI Auto Logon Recording Options

If you are recording a multi-protocol Vuser script with a SAP-Web Vuser type, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168 for additional recording options.

SAPGUI General Recording Options

You use these recording options to set your general preferences during the recording session in the following areas:



- **Capture Screen Snapshots.** Indicates how to save the snapshots of the SAPGUI screens as they appear during recording: **ActiveScreen snapshots**, **Regular snapshots**, or **None.** ActiveScreen snapshots provide more interactivity and screen information after recording, but they require more resources.

➤ **Process Context menus by text.** Instructs VuGen to process context menus by their text, generating a **sapgui_toolbar_select_context_menu_item_by_text** function. When disabled, VuGen processes context menus by their IDs, an advantage when working with Japanese characters. In the latter case, VuGen generates a **sapgui_toolbar_select_context_menu_item** for context menus.

To set the General recording options:

1 Open the Recording Options dialog box and select the **SAPGUI:General** node.

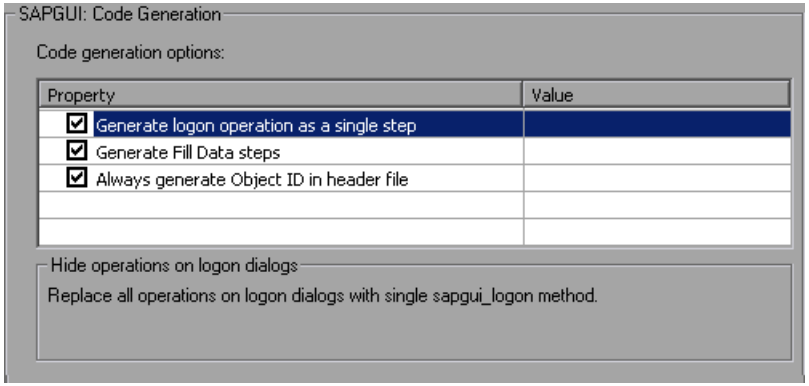
2 For the **Capture screen snapshots** option, indicate how to save the snapshots of the SAPGUI screens.

3 Select the method by which to process context menus: Enable **Process context menus by text**, or disable it to process context menus by their IDs.

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

SAPGUI Code Generation Recording Options

The following recording options indicate your code generation preferences.

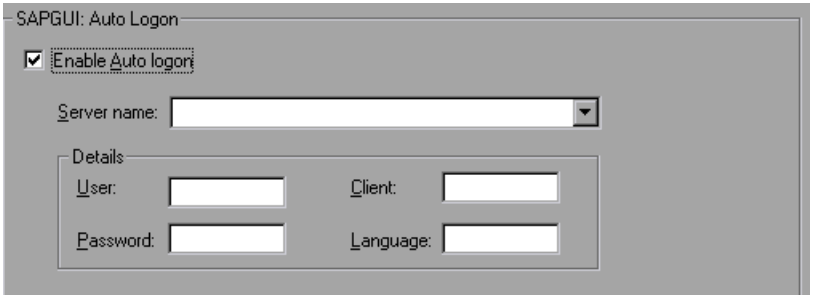


To set the Code Generation recording options:

1. Open the Recording Options dialog box and select the **SAPGUI:Code Generation** node.
2. Select **Generate logon operation as a single step** to instruct VuGen to generate a single **sapgui_logon** method for all of the logon operations. This helps simplify the code. If you encounter login problems, disable this option.
3. To generate Fill Data steps for table and grid controls—instead of separate steps for each cell, select **Generate Fill Data Steps**.
4. To create a more compact and cleaner script, select **Always generate Object IDs in header file** which places the Object IDs in a separate header file instead of in the script. When you disable this option, VuGen generates the IDs according to the specified string length in the general script setting. Note that disabling this option only increases readability—there is no difference in overhead.
5. Click **OK** to accept the settings and close the dialog box.

SAPGUI Auto Logon Recording Options

You set these recording options to log on automatically when you begin recording. The logon functions are placed in the vuser_init section of the script. The server name list contains all of the servers on the SAP Logon description list



To enable and set the Auto Logon recording options:

1 Open the Recording Options dialog box and select the **SAPGUI:Auto Logon** node.

2 Select **Enable Auto logon**.

3 Enter the Login information:

- the SAP **Server name**
- the **User** name for the SAP server
- the **Password** for the SAP server
- the **Client** name by which the SAP server identifies the client
- the interface **Language**

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

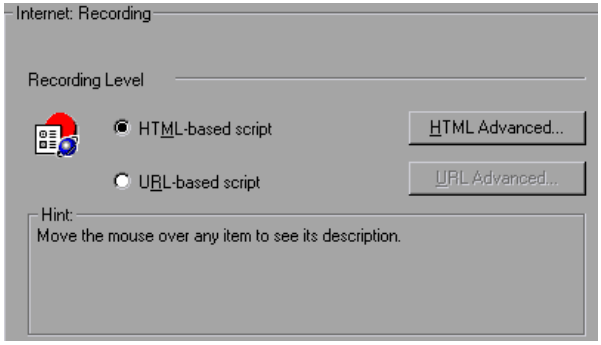
SAP-Web Recording Options

You use the recording options to set your preferences for how VuGen generates the Vuser script.

The recommended settings for the **General:Recording** node are:

For SAP Workplace recordings: URL-based script

For SAP Portal recordings: HTML-based script (the default)



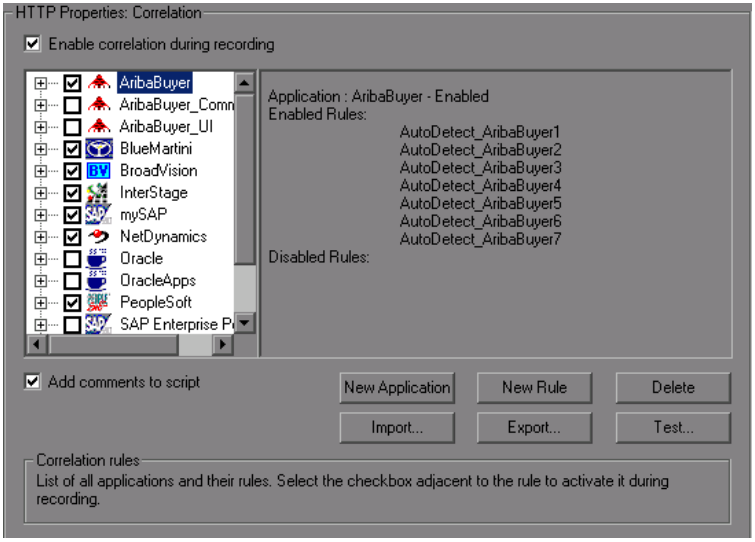
For information about the other Web related recording options, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168

Setting the Correlation Recording Options

To instruct VuGen to correlate your statements during recording, you set the Correlation recording options. You set these options after opening a Web Vuser script but before you begin recording the session.

To set the correlation recording options:

1 After you create a script, but before you begin recording, select **Tools > Recording Options** and select the **HTTP Properties:Correlation** node in the Recording Options tree.



2 Select the **Enable correlation during recording** option.

3. Indicate the servers to which you want to apply the correlation rules. Select the check boxes adjacent to the server names to enable the rules for that server. To enable specific rules within a server group, click the plus sign to expand the tree and select the desired rules.
4. To add a new rule to an existing server, select one of the existing entries and click **New Rule**. Set the properties for the rule in the right pane. For more information, see "Setting Correlation Rules" on page 859.
5. To add a set of rules for a new application, click **New Application**. Then click **New Rule** to create a rule for the application.
6. To modify the properties of an existing rule, select the rule in the left pane and modify the rules in the right pane.
7. To delete an application or rule, select it and click **Delete**. VuGen prompts you to confirm your choice before deleting the selection.
8. To export a set of correlation rules, click **Export** and save the **.cor** file to the desired location. To import a set of correlation rules created during an earlier session, click **Import** and open the file from its location.
9. Click **OK**.

Web, Wireless, and Oracle NCA Recording Options

Use the **HTTP Properties:Advanced** settings to set the recording options in the following areas:

- Internet Preferences Recording Options
- Selecting a Recording Engine
- Setting a Recording Scheme

Internet Preferences Recording Options

The Internet Preference options allow the customization of code generation settings in the area of think time, resetting contexts, saving snapshots, and the generation of **web_reg_find** functions. Note that some of these options are not available in multi-protocol mode.

- **Record think time.** (Wireless only) This setting, enabled by default, tells VuGen to record the think times and generate **lr_think_time** functions. You can also set a **Think-time Threshold** value—if the actual think-time is less than the threshold, VuGen does not generate a **lr_think_time** function.
- **Reset context for each action.** (Web, Oracle NCA only) This setting, enabled by default, tells VuGen to reset all HTTP contexts between actions. Resetting contexts allows the Vuser to more accurately emulate a new user beginning a browsing session. This option resets the HTML context, so that a contextless function is always recorded in the beginning of the action. It also clears the cache and resets the user names and passwords.
- **Save snapshot resources locally.** This option instructs VuGen to save a local copy of the snapshot resources during record and replay. This feature lets VuGen create snapshots more accurately and display them quicker.
- **Generate web_reg_find functions for page titles.** (Web, Oracle NCA only) This option enables the generation of **web_reg_find** functions for all HTML page titles. VuGen adds the string from the page's title tag and uses it as an argument for **web_reg_find**.

- **Generate web_reg_find functions for sub-frames.** Enables the generation of **web_reg_find** functions for page titles in all sub-frames of the recorded page.

- **Add comment to script for HTTP errors while recording.** This option adds a comment to the script for each HTTP request error. An error request is defined as one that generated a server response value of 400 or greater during recording.

- **Support charset.**
  - **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. You should enable this option only on non-English UTF-8 encoded pages. The recorded site's language must match the operating system language. You cannot record non-English Web pages with different encodings (for example, UTF-8 together with ISO-8859-1 or shift_jis) within the same script.
  - **EUC-JP.** 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's machine, and adds a **web_sjis_to_euc_param** function to the script. (Kanji only)

Selecting a Recording Engine

By default, for Web(HTTP/HTML) Vusers, VuGen uses the multi-protocol recording engine for all recordings, even if you are only recording a single protocol.

To use the single-protocol recording engine for backward compatibility, select the **Record script using earlier recording engine** option in the Advanced Recording Options. If you enable this option, VuGen will use the single-protocol engine the next time you record a Web(HTTP/HTML) session.

Setting a Recording Scheme

You can further customize the recording by specifying a recording scheme in the following areas:

- Recording Custom Headers
- Filtering Content Type
- Specifying Non-Resource Content Types

Recording Custom Headers

Web Vusers automatically send several standard HTTP headers with every HTTP request submitted to the server. Click **Headers** to instruct VuGen to record additional HTTP headers. You can work in three modes: **Do not Record headers**, **Record headers in list,** or **Record headers not in list.** When you work in the first mode, VuGen does not record any headers. In the second mode, VuGen only records the checked custom headers. If you specify **Record headers not in list,** VuGen records all custom headers except for those that are checked and other risky headers.

The following standard headers are known as **risky** headers: Authorization, Connection, Content-Length, Cookie, Host, If-Modified-Since, Proxy-Authenticate, Proxy-Authorization, Proxy-Connection, Referer, and WWW-Authenticate. They are not recorded unless selected in the Header list. The default option is **Do not record headers**.

In the **Record headers in list** mode, VuGen inserts a **web_add_auto_header** function into your script for each of the checked headers that it detects. This mode is ideal for recording risky headers that are not recorded unless explicitly stated.

In the **Record headers not in list** mode, VuGen inserts a **web_add_auto_header** function into your script for each of the unchecked headers that it detects during recording.

To determine which custom headers to record, you can perform a recording session indicating to VuGen to record all headers (see procedure below). Afterwards, you can decide which headers to record and which to exclude.

In this example, the Content-type header was specified in the Record headers in list mode. VuGen detected the header and added the following statement to the script:

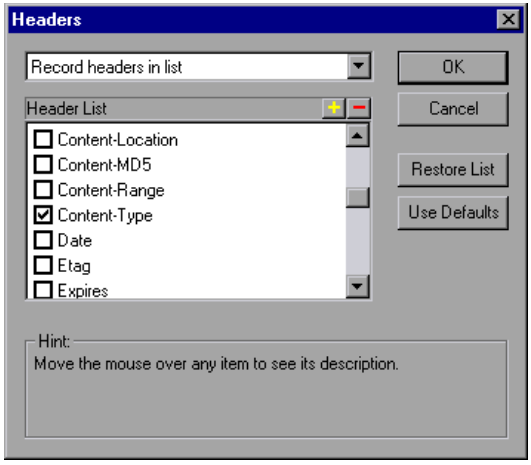
```
web_add_auto_header("Content-Type",
                    "application/x-www-form-urlencoded");
```

indicating to the server that the Content-type of the application is x-www-form-urlencode.

To control the recording of custom headers:

1 In the Recording Options tree, select the **HTTP Properties:Advanced** node.

2 Click **Headers.** The Headers dialog box opens.



3 Use one of the following methods:

- To instruct VuGen not to record any Headers, select **Do not record headers**.
- To record only specific headers, select **Record headers in list** and select the desired custom headers in the header list. Note that standard headers (such as **Accept**), are selected by default.
- To record all headers, select **Record headers not in list** and do not select any items in the list.

➤ To exclude only specific headers, select **Record headers not in list** and select the headers you want to exclude.

4 Click **Restore List** to restore the list to the corresponding default list. The **Record headers in list** and **Record headers not in list** each have a corresponding default list.

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

Filtering Content Type

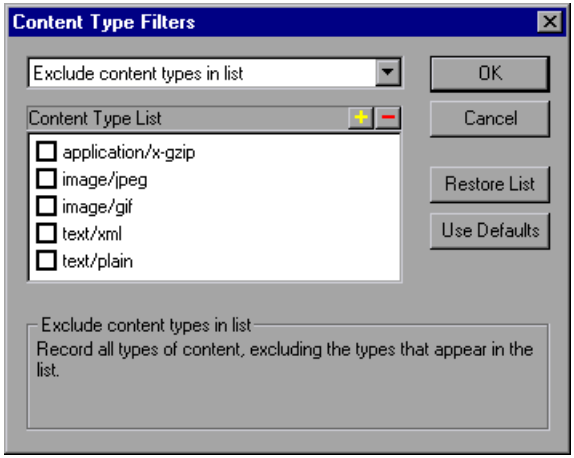
VuGen allows you to filter the content type for your recorded script. You specify the type of the content you want to record or exclude from your script. You can work in three modes: **Do not filter content types**, **Exclude content types in list,** or **Exclude content types not in list.** When you work in the first mode, VuGen does not filter any content type. In the second mode, VuGen only excludes the selected content types. If you specify **Exclude content types not in list,** VuGen filters all content type except for the ones that are checked. By default, no filters are active.

For example, if you are only interested in the text and images on your Web site, you select **Exclude content types not in list** and specify the types **text/html**, **image/gif**, and **image/jpeg**. VuGen will record all HTML pages and images, and exclude resources such as **text/css**, **application/x-javascript**, or other resources that appear on the site.

To filter content during recording:

1 In the Recording Options tree, select the **HTTP Properties:Advanced** node.

2 Click **Content Types.** The Content Type Filters dialog box opens.



3 Use one of the following methods:

- To instruct VuGen not to filter any content, select **Do not filter content types**.
- To exclude only specific content types, select **Exclude content types in list** and select the desired content types from the list.
- To include only specific content types, select **Exclude content types not in list** and select the content types you want to include.

4 Click **Restore List** to restore the list to the corresponding default list. The **Exclude content types in list** and **Exclude content types not in list** each have a corresponding default list.

5 Click **OK** to accept the settings and close the Content Type Filters dialog box.

Specifying Non-Resource Content Types

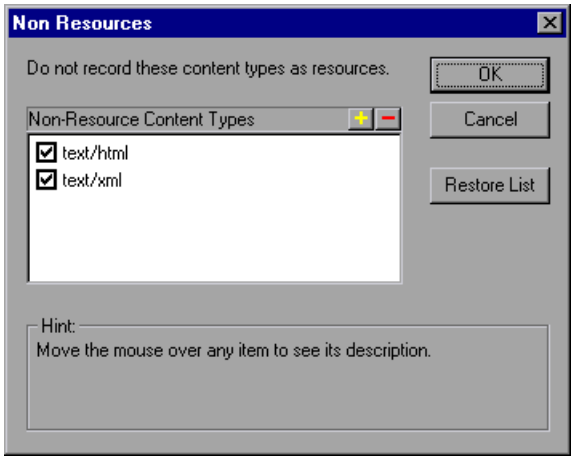
When you record a script, VuGen indicates whether or not it will retrieve the resource during replay using the Resource attribute in the web_url function. If the Resource attribute is set to 0, the resource is retrieved during script execution. If the Resource attribute is set to 1, the Vuser skips the resource type.

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

You can exclude specific content types from being handled as resources. For example, you can indicate to VuGen that **gif** type resources should not be handled as a resource and therefore be downloaded unconditionally. When VuGen encounters a **gif** type resource, it sets the **Resource** attribute to 0, indicating to VuGen to download gifs unconditionally during replay.

To specify which content should not be recorded as resources:

1. In the Recording Options tree, select the **HTTP Properties:Advanced** node.
2. Click **Non-Resources** to open the dialog box and display the list of content types which should not be recorded as resources.



3. Click the "+" sign to add a content type to the list. Click the "-" sign to remove an existing entry.
4. Select the check boxes adjacent to the items you want to enable.
5. Click **Restore List** to restore the list to the default list.
6. Click **OK** to accept the settings and close the Non-Resources list.

74

Click and Script Recording

The Click and Script solution lets you record Web sessions on a user-action. The following chapter contains information about Click and Script recording and recording options.

This chapter includes:

The following information applies to the AJAX (Click and Script), Web (Click and Script), SAP (Click and Script), Oracle Web Applications 11i, and PeopleSoft Enterprise protocols.

About Recording with Click and Script

The Click and Script mechanism lets you record Web sessions on a user-action GUI level. VuGen creates a GUI-level script that intuitively represents actions in the Web interface. For example, it generates a **web_button** function when you click a button to submit information, and generates a **web_edit_field** function when you enter text into an edit box. For SAP applications, VuGen records a **sap_button** function.

For pure Web sessions, the Web (HTTP/HTML) Vuser type create a lower level script. For more information about choosing a script type for your Web session, see "Selecting a Web Vuser Type" on page 773.

Viewing Web (Click and Script) Vuser Scripts

Click and Script Vuser scripts typically contain several actions which make up a business process. By viewing the recorded functions that were generated on a GUI level, you can determine the user's exact actions during the recorded session.

For example, in a typical recording, the first stage may contain a sign-in process. The browser opens on the sign-in page, and a user signs in by submitting a user name and password and clicking **Sign In**.

For the Web (Click and Script) Vuser, VuGen generates a **web_edit_field** function that represents the data entered into an edit field. In the example that follows, a user entered text into the userid field, and a password into the pwd field which is encrypted.

```
vuser_init()
{
    web_browser("WebTours",
        DESCRIPTION,
        ACTION,
        "Navigate=http://localhost:1080/WebTours/",
        LAST);

    web_edit_field("username",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=text",
        "Name=username",
        "FrameName=navbar",
        ACTION,
        "SetValue=jojo",
        LAST);

    web_edit_field("password",
        "Snapshot=t3.inf",
        DESCRIPTION,
        "Type=password",
        "Name=password",
        "FrameName=navbar",
        ACTION,
        "SetEncryptedValue=440315c7c093c20e",
        LAST);…
```

Setting Click and Script Recording Options

Before recording a script, you can set options that indicate what to record and how to generate the script after the recording.

You can set common recording options in the following areas: General, HTTP Properties, and Network.

The following sections discuss the GUI Properties recording options that are specific for all Vuser types that use the Click and Script mechanism: AJAX (Click and Script), Web (Click and Script), SAP (Click and Script), Oracle Applications, and PeopleSoft Enterprise Vusers. These recording options let you indicate the events to be recorded and which properties to include for each object.

- Setting Advanced GUI Properties
- Configuring Web Event Recording

For information on the other Recording Options, see the appropriate section:

- **General: Script.** See Chapter 76, "Setting Script Generation Preferences."
- **General: Recording.** See Chapter 77, "Setting Recording Options for Web Vusers."
- **Network: Port Mapping.** See "Configuring the Port Mappings" on page 1233
- **HTTP Properties: Advanced.** See "Web, Wireless, and Oracle NCA Recording Options" on page 1168. Note that the **Save snapshot resources locally** and **Generate web_reg_find functions for page titles** options do not apply to GUI-based scripts (see explanation of GUI-based scripts below).
- **HTTP Properties: Correlation.** See Chapter 53, "Web (HTTP/HTML) Correlation Rules". Note that there are built-in rules for the Oracle and PeopleSoft servers. To enable them, select the check box adjacent to your server name.

Several additional HTTP properties are only configurable for Web (HTTP/HTML) scripts.

Setting Advanced GUI Properties

VuGen lets you set Click and Script advanced options in the following areas:

- Recording Settings
- Code Generation Settings

Recording Settings

The Recording settings instruct VuGen what to record. You can enable or disable the following features:

Record rendering-related property values

Records the values of the rendering-related properties of DOM objects (for example, **offsetTop**), so that they can be used during replay. Note that this may significantly decrease the replay speed (disabled by default).

Record 'click' by mouse events

Records mouse clicks by capturing mouse events instead of capturing the click() method. Enable when the recorded application uses the DOM click() method, to prevent the generation of multiple functions for the same user action (enabled by default).

Record socket level data

Enables the recording of socket level data. If you disable this option you will need to manually add the starting URL before recording. In addition, you will be unable to regenerate the script on an HTML level. (enabled by default).

Generate snapshots for AJAX steps

Enables generation of snapshots for AJAX steps. This is unchecked by default. Checking this option can result in errors during recording.

Code Generation Settings

The Code Generation settings instruct VuGen how to generate the script after the recording. You can enable or disable the following features:

Enable generation of out-of-context steps

You can instruct VuGen to create a URL-based script for ActiveX controls and Java applets, so that they will be replayed. Since these functions are not part of the native recording, they are referred to as out-of-context recording (disabled by default).

In the following example, the script was regenerated with the out-of-context recording option enabled.

```
web_image_link("Search Flights Button",
        "Snapshot=t5.inf",
        DESCRIPTION,
        "Alt=Search Flights Button",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=58,9",
        LAST);

web_add_cookie("MSO=SID&1141052844; DOMAIN=localhost");

web_add_cookie("MTUserInfo=hash&47&firstName&Joseph&expDate&%0A&creditCa
rd&&address1&234%20Willow%20Drive&lastName&Marshall%0A&address2&San%2
0Jose%2FCA%2F94085&username&jojo; DOMAIN=localhost");

web_url("FormDateUpdate.class",
        "URL=http://localhost:1080/WebTours/FormDateUpdate.class",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "UserAgent=Mozilla/4.0 (Windows 2000 5.0) Java/1.4.2_08",
        "Mode=HTTP",
        LAST);
…
```

If you disable this option, VuGen does not generate code for the ActiveX controls and Java applets. In the following example, VuGen only generated the **web_image_link** function—not the **web_url** functions containing the class files.

```
web_image_link("Search Flights Button",
        "Snapshot=t5.inf",
        DESCRIPTION,
        "Alt=Search Flights Button",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=58,9",
        LAST);
```

Enable automatic browser title verification

Enables automatic browser title verification (disabled by default).

You can also customize the type of title verification.

- **Perform a title verification for.**

  each navigation. Performs a title verification only after a navigation. When a user performs several operations on the same page, such as filling out a multi-field form, the title remains the same and verification is not required.

  each step. Performs a title verification for each step to make sure that no step modified the browser title. A modified browser title may cause the script to fail.

- **Perform a title verification using the URL if the title is missing.** For browser windows without a title, perform a title verification for each step using its URL.

Configuring Web Event Recording

VuGen creates a script by recording HTML object events triggered by user actions, such as clicking the mouse or pressing a key while viewing the document.

You may find that you need to record more or fewer events than VuGen automatically records by default. You can modify the default event recording settings by using the Web Event Recording Configuration dialog box to select one of three standard configurations, or you can customize the individual event recording configuration settings to meet your specific needs.

This section describes how to configure VuGen's handling of Web Events:

- Selecting a Standard Event Recording Configuration
- Customizing the Event Recording Configuration
- Adding and Deleting Listening Events for an Object
- Modifying the Listening and Recording Settings for an Event
- Resetting Event Recording Configuration Settings

For example, VuGen does not generally record mouseover events on link objects. If, however, you have a mouseover handler connected to a link, it may be important for you to record the mouseover event. In this case, you could customize the configuration to record mouseover events on link objects whenever they are connected to a handler.

Note: Event configuration is a global setting and therefore affects all tests that are recorded after you change the settings.

Changing the event configuration settings does not affect tests that have already been recorded. If you find that VuGen recorded more or less than you need, change the event recording configuration and then re-record the part of your test that is affected by the change.

Changes to the custom Web event recording configuration settings do not take effect on open browsers. To apply your changes for an existing test, make the changes you need in the Web Event Recording Configuration dialog box, refresh any open browsers, and then start a new recording session.

Selecting a Standard Event Recording Configuration

The Web Event Recording Configuration dialog box offers three standard event configuration levels, Basic, Medium, or High. By default, VuGen uses the Basic configuration level. If VuGen does not record all the events you need, you may require a higher event configuration level.

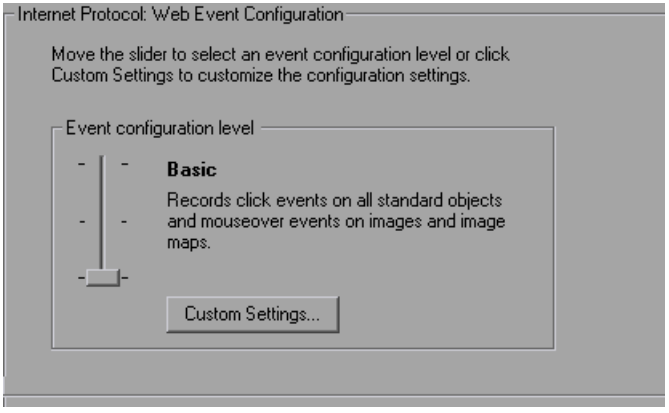
Level	Description
Basic	**Default** ➤ Always records click events on standard Web objects such as images, buttons, and radio buttons. ➤ Always records the submit event within forms. ➤ Records click events on other objects with a handler or behavior connected. For more information on handlers and behaviors, see "Listening Criteria" on page 1191. ➤ Records the mouseover event on images and image maps only if the event following the mouseover is performed on the same object.

Level	Description
Medium	In addition to the objects recorded in the **Basic** level, it records click events on the <DIV>, <SPAN>, and <TD> HTML tag objects.
High	In addition to the objects recorded in the **Medium** level, it records mouseover, mousedown, and double-click events on objects with handlers or behaviors attached. For more information on handlers and behaviors, see "Listening Criteria" on page 1191.

To set a standard event-recording configuration:

1 Open the Recording Options dialog box. Select **Tools** > **Recording Options**.

2 Select the **GUI Properties:Web Event Configuration** node.



3 Use the slider to select your preferred standard event recording configuration: **Basic**, **Medium**, or **High**.

Tip: Click the **Custom Settings** button to open the Custom Web Event Recording dialog box where you can customize the event recording configuration. For more information, see "Customizing the Event Recording Configuration" below.

4 Click **OK**.

Customizing the Event Recording Configuration

If the standard event configuration levels do not exactly match your recording needs, you can customize the event recording configuration using the Custom Web Event Recording Configuration dialog box.

You can customize Web events for standard Web elements, such as an image, link, WebArea, WebButton, and so forth. You can also set the recording behavior for any HTML tag that you choose. You add the desired HTML tag object and then set its recording behavior. For example, you can configure VuGen to record all **mouseover** events for all **DIV** tags.

The Custom Web Event Recording Configuration dialog box enables you to customize event recording in several ways. You can:

- Enable or disable objects to which VuGen should apply special listening or recording settings
- Add or remove events for which VuGen should listen
- Modify the listening and recording settings for an event

You can modify the event recording configuration using the following options:

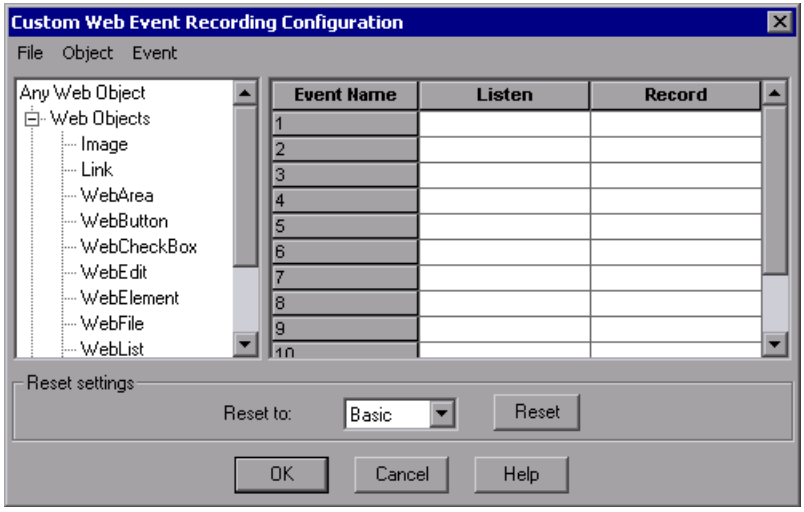
Option	Description
Event Name	Displays a list of events associated with the object. ➤ To add an event to the Events pane, select **Event** > **Add**. Select the desired event. ➤ To delete an event, click on the event in the Event Name columns, and select **Event** > **Delete**. For more information, see "Adding and Deleting Listening Events for an Object" on page 1190.
Listen	The criteria for when VuGen listens to the event. ➤ **Always.** Always listens to the event. ➤ **If Handler.** Listens to the event if a handler is attached to it. A handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs. ➤ **If Behavior.** Listens to the event if a DHTML behavior is attached to it. A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior. ➤ **If Handler or Behavior.** Listens to the event if a handler or behavior is attached to it. ➤ **Never.** Never listens to the event. For more information, see "Modifying the Listening and Recording Settings for an Event" on page 1191.
Record	Enables or disables recording of the event for the selected object, or enables recording of the event only if the subsequent event occurs on the same object.
Reset	Resets your settings to a pre-configured level: **Basic**, **Medium**, or **High**.

To customize the recording configuration for an event:

1 Open the Recording Options dialog box. Select **Tools** > **Recording Options**.

2 Select the **GUI Properties:Web Event Configuration** node.

3 Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.



4 Specify an object:

- To configure one of the built-in Web objects, select it in the left pane.
- To specify a different HTML tag object, select **Object** > **Add** and enter the name of the HTML tag whose event you want to record.

5 In the right pane, specify the **Listen** and **Record** behavior for each event as described above. If the event you want to record does not appear in the right pane, select **Event** > **Add** to add the event, as described below.

6 Click **OK** to save the customization.

7 Click **Reset** to reset your settings to a pre-configured level: **Basic**, **Medium**, or **High**.

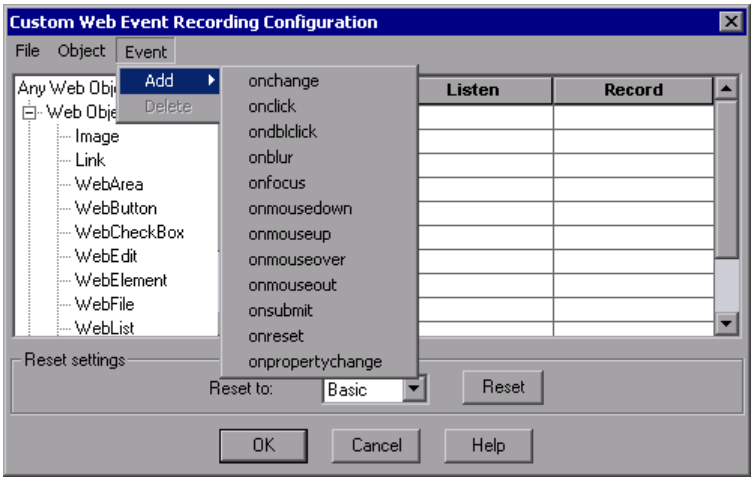
Adding and Deleting Listening Events for an Object

You can modify the list of events that trigger VuGen to listen to an object.

To add listening events for an object:

1 In the Custom Web Event Recording Configuration dialog box, select the object to which you want to add an event—one of the built-in Web objects or an HTML tag object.

2 Select **Event** > **Add**. A list of available events opens.



3 Select the event you want to add. The event is displayed in the Event Name column in alphabetical order. By default, VuGen listens to the event when a handler is attached and always records the event (as long as it is listened to at some level).

For more information on listening and recording settings, see "Modifying the Listening and Recording Settings for an Event" below.

To delete listening events for an object:

1 In the Custom Web Event Recording Configuration dialog box, select the object from which you want delete an event in the left pane.

2 In the **Event Name** column, select the event you want to delete.

3 Select **Event** > **Delete**. The event is deleted from the **Event Name** column.

Modifying the Listening and Recording Settings for an Event

You can select the listening criteria and set the recording status for each event listed for each object.

Note: The listen and record settings are mutually independent. This means that you can listen to an event for a particular object, but not record it, or you can choose not to listen to an event for an object, but still record the event. For more information, see "Tips for Working with Event Listening and Recording" on page 1193.

Listening Criteria

For each event, you can instruct VuGen when to listen for an event:

- **Always.** Listen every time the event occurs on the object.
- **If Handler.** Listen only if an event handler is attached to the event.
- **If Behavior.** Listen only if a DHTML behavior is attached to the event.
- **If Handler or Behavior.** Listen if an event handler or a DHTML behavior is attached to the event.
- **Never.** Never listen to the event.

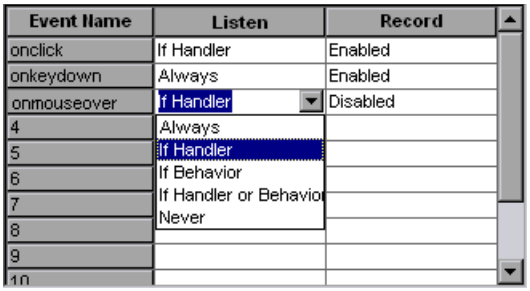
An event **handler** is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs.

A DHTML **behavior** encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior.

To specify the listening criterion for an event:

1 From the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the listening criterion.

2 In the row of the event you want to modify, select the listening criterion you want from the **Listen** column.



Select a listening criteria from the list: **Always, If Handler**, **If Behavior**, **If Handler or Behavior**, or **Never**.

Recording Status

For each event, you can enable recording, disable recording, or enable recording only if the next event is dependent on the selected event.

➤ **Enabled.** Records the event each time it occurs on the object as long as VuGen listens to the event on the selected object, or on another object to which the event bubbles.

Bubbling is the process whereby, when an event occurs on a child object, the event can travel up the chain of hierarchy within the HTML code until it encounters an event handler to process the event.

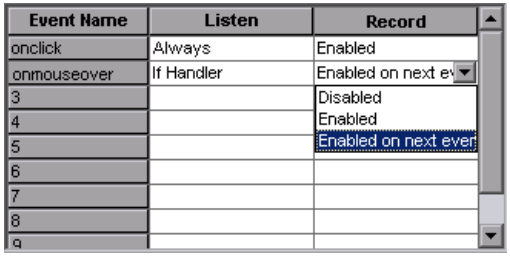
➤ **Disabled.** Does not record the specified event and ignores event bubbling where applicable.

➤ **Enabled on next event.** (only applicable to the Image and WebArea objects) Same as **Enabled**, except that it records the event only if the subsequent event occurs on the same object. For example, suppose a mouseover behavior modifies an image link. You may not want to record the mouseover event each time you happen to move the mouse over this image. Because only the image that is displayed after the mouseover event enables the link event, however, it is essential that the mouseover event is recorded before a click event on the same object.

To set the recording status for an event:

1 In the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the recording status. Select **Any Web Object** to set the recording status for all Web objects in the recorded pages.

2 In the row of the event you want to modify, select a recording status from the Record column.



Tips for Working with Event Listening and Recording

It can sometimes be difficult to find the ideal listen and recording settings. When defining these settings, keep in mind the following guidelines:

➤ To record an event on an object, you must instruct VuGen to listen for the event, and to record the event when it occurs. You can listen for an event on a child object, even if a parent object contains the handler or behavior, or you can listen for an event on a parent object, even if the child object contains the handler or behavior.

However, you must enable recording for the event on the source object (the one on which the event actually occurs, regardless of which parent object contains the handler or behavior).

For example, suppose a table cell with an **onmouseover** event handler contains two images. When a user touches either of the images with the mouse pointer, the event also bubbles up to the cell, and the bubbling includes information on which image was actually touched. You can record this mouseover event by:

- Setting **Listen** on the WebTable mouseover event to **If Handler** (so that VuGen "hears" the event when it occurs), while disabling recording on it, and then setting **Listen** on the Image mouseover event to **Never**, while setting its recording status to **Enable** (to record the mouseover event on the image after it is listened to at the WebTable level).
- Setting **Listen** on the Image mouseover event to **Always** (to listen for the mouseover event even though the image tag does not contain a behavior or handler), and setting the recording status on the Image object to **Enabled** (to record the mouseover event on the image).

- Instructing VuGen to listen for many events on many objects may lower performance, so try to limit listening settings to the required objects.
- In rare situations, listening to the object on which the event occurs (the source object) may interfere with the event.

Resetting Event Recording Configuration Settings

After you set custom settings, you can restore standard settings by instructing VuGen to use the default Web Event configuration settings.

Note: When you reset standard settings, your custom settings are cleared completely.

To restore basic level configuration settings:

1. In the Recording Options, select the **GUI Properties:Web Event Configuration** node.
2. Click **Use Defaults**. The standard configuration slider is displayed again and all event settings are restored to the **Basic** event recording configuration level.

You can also restore the settings to a specific (base) custom configuration: Basic, Medium, or High.

To reset the settings to a custom level:

1. In the Recording Options, select the **GUI Properties:Web Event Configuration** node.
2. Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.
3. In the **Reset to** box, select the standard event recording level you want.
4. Click **Reset**. All event settings are restored to the defaults for the level you selected.

75

Java Recording Options

VuGen allows you to control the way in which you record your CORBA, RMI, JMS or Jacada application. You can use the default recording options, or customize them for your specific needs.

This chapter includes:

About Setting Java Recording Options

Using VuGen, you record a CORBA (Common Object Request Broker Architecture) or RMI (Remote Method Invocation) Java application or applet. For recording an EJB test, see Chapter 28, "Enterprise Java Beans (EJB) Protocol".

Before recording, VuGen lets you set recording options for the Java Virtual Machine (JVM) and for the code generation stage. Setting the recording options is not mandatory; if you do not set them, VuGen uses the default values.

The options described in this chapter were previously handled by modifying the **mercury.properties** file.

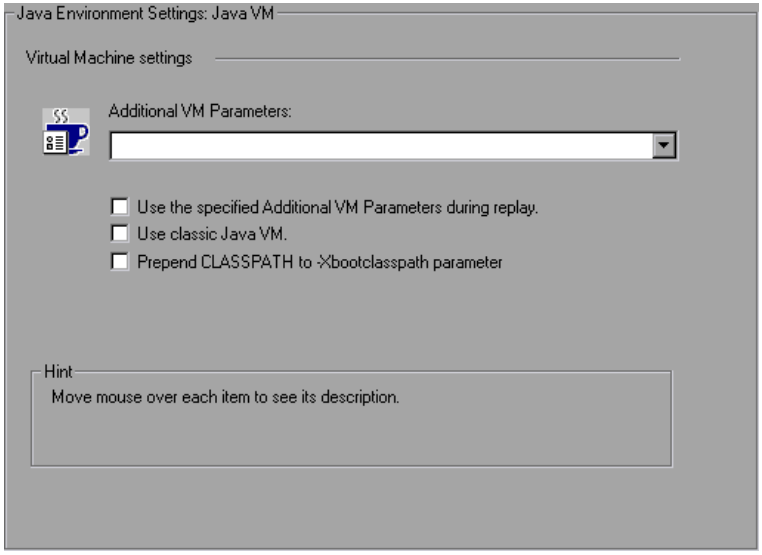
You can set recording options in the following areas:

- Java Virtual Machine (JVM) Recording Options
- Setting Classpath Recording Options
- Recorder Options
- Serialization Options
- Correlation Options
- Log Options

Java Virtual Machine (JVM) Recording Options

The **Java VM** options indicate additional parameters to use when recording Java applications.

When you record a Vuser, VuGen automatically sets the **Xbootclasspath** variable with default parameters. If you use this dialog box to set the **Xbootclasspath** with different parameters, it will use those command parameters—not the default ones.



You can also instruct VuGen to add the Classpath before the **Xbootclasspath** (prepend the string) to create a single Classpath string.

By default, VuGen uses the classic VM during recording. You can also instruct VuGen to use another virtual machine (Sun's Java Hotspot VM).

To set the Java Virtual Machine recording options:

1. Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Java VM** node in the Recording Options tree.

2. In the **Additional VM Parameters** box, list the Java command line parameters. These parameters may be any Java VM argument. The common arguments are the debug flag (**-verbose**) or memory settings (**-ms**, **-mx**). For more information about the Java VM flags, see the JVM documentation. In additional, you may also pass properties to Java applications in the form of a **-D** flag.

   VuGen automatically sets the **-Xbootclasspath** variable (for JDK 1.2 and higher) with default parameters. When you specify **-Xbootclasspath** with parameter values as an additional parameter, VuGen uses this setting instead of the default one.

3. To use the same Additional VM parameters in replay, select the **Use the specified Additional VM Parameters during replay** check box.

4. To use the classic VM, select the **Use classic Java VM** check box (default). To use another VM (Sun's Java HotSpot), clear the check box.

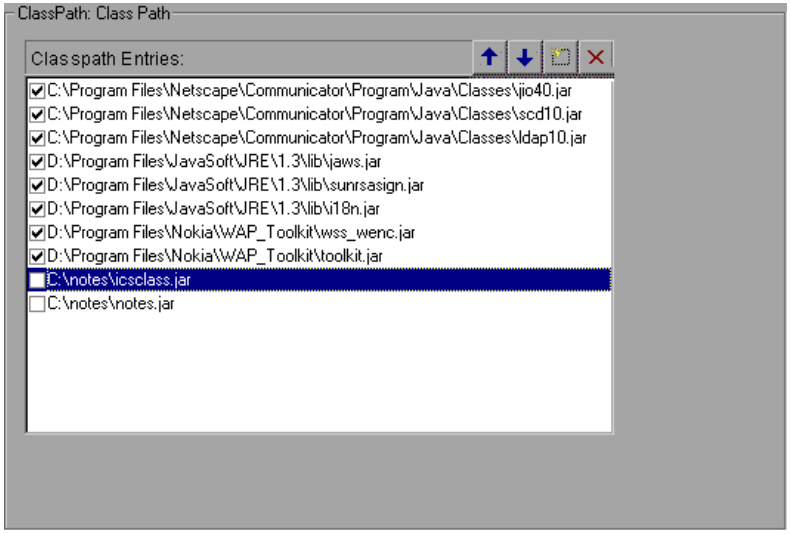
5. To add the Classpath before the **Xbootclasspath** (prepend the string), select the **Prepend CLASSPATH to -Xbootclasspath parameter** check box.

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

Setting Classpath Recording Options

The **Java Environment Settings**:**Classpath** node lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper recording.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.



To set the Classpath recording options:

1 Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Classpath** node in the Recording Options tree.

2 To add a classpath to the list:

Click the **Add Classpath** button. VuGen adds a new line to the classpath list.

Type in the path and name of the **jar**, **zip** or other archive file for your class. Alternatively, click the **Browse** button to the right of the field, and locate the desired file. VuGen adds the new location to the classpath list, with an enabled status.

3 To permanently remove an entry, select it and click the Delete button.

4 To disable a classpath entry for a specific test, clear the check box to the left of the entry.

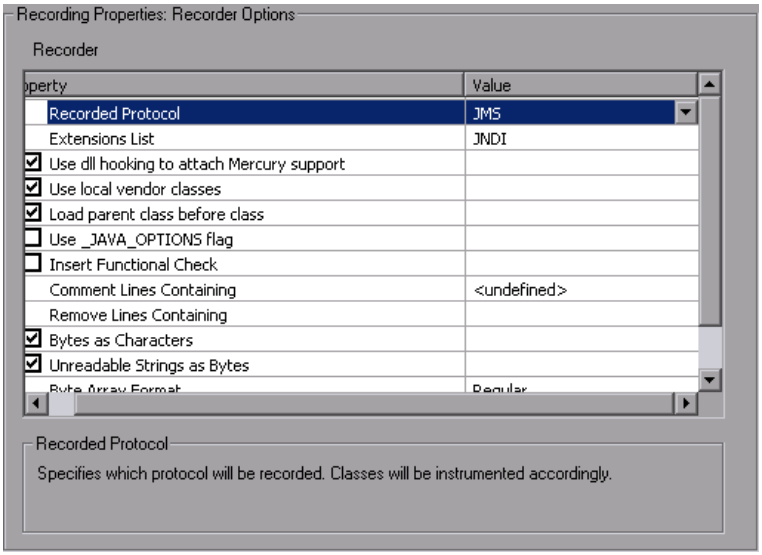
5 To move an entry down in the list, select it and click the Down arrow.

6 To move a classpath entry up within the list, select it and click the Up arrow.

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

Recorder Options

The **Recorder** options indicate which protocol to record and some of the protocol-specific settings.



- **Recorded protocol.** Specifies which protocol to record: RMI, CORBA, JMS, or Jacada. (RMI by default).
- **Extensions list.** A comma separated list of all supported extensions. Each extension has its own hooks file (JNDI by default).

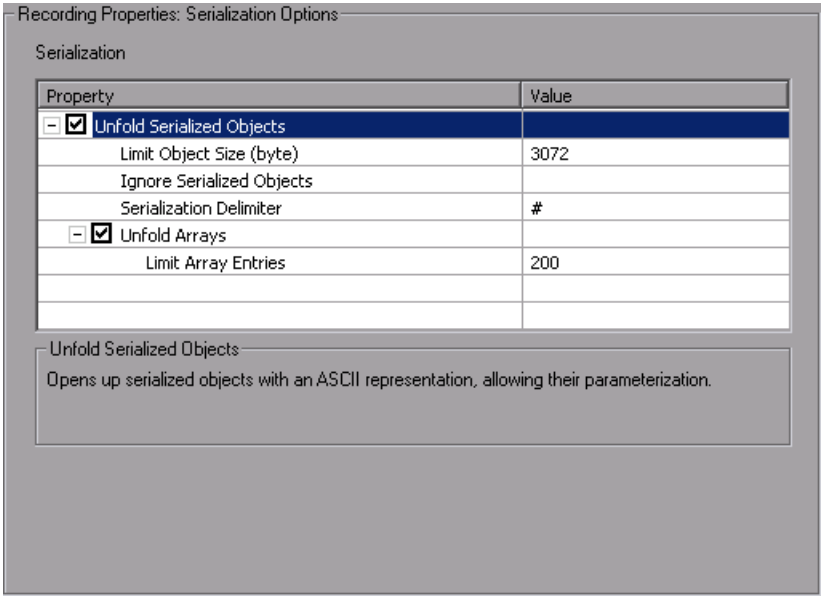
- **Use DLL hooking to attach LoadRunner support.** Use DLL hooking to automatically attach LoadRunner support to any JVM.
- **Load parent class before class.** Change the loading order so that parent classed are loaded before child classes. This helps identify hooking for trees with deep inheritance. (enabled by default).
- **Use _JAVA_OPTION flag.** Forces JVM versions 1.2 and higher to use the _JAVA_OPTION environment variable which contains the desired JVM parameters (disabled by default).
- **Insert functional check.** Inserts verification code that compares the return value received during replay, to the expected return value generated during recording. This option only applies to primitive return values (disabled by default).
- **Comment lines containing.** Comment out all lines in the script containing one of the specified strings. To specify multiple strings, separate the entries with commas. By default, any line with a string containing <undefined>, will be commented out.
- **Remove lines containing.** Remove all lines containing one of the specified strings from the script. To specify multiple strings, separate the entries with commas. This feature is useful for customizing the script for a specific testing goal.
- **Bytes as characters.** Displays readable characters as characters with the necessary casting—not in byte or hexadecimal form (enabled by default).
- **Unreadable strings as bytes.** Represents strings containing unreadable characters as byte arrays. This option applies to strings that are passed as parameters to invocations (enabled by default).
- **Byte array format.** The format of byte arrays in a script: **Regular**, **Unfolded Serialized Objects**, or **Folded Serialized Objects**. Use one of the serialized object options when recording very long byte arrays. The default is **Regular**.
- **Record LoadRunner callback.** Records the LoadRunner stub object as a callback. If disabled, VuGen records the original class as the callback (enabled by default).

To set the Java Recorder options:

1. Click **Options** in the Start Recording dialog box and select the **Recording Properties:Recorder Options** node.
2. Set the options as desired. For the options with check boxes, select or clear the check box adjacent to the option. For options that require strings, type in the desired value.
3. To set all options to their default values, click **Use Defaults**.
4. Click **OK** to close the dialog box and begin recording.

Serialization Options

The **Serialization** options let you to control how objects are serialized. Serialization is often relevant to displaying objects in an ASCII representation in order to parameterize their values.



The following options are available:

- **Unfold Serialized Objects.** Expands serialized objects in ASCII representation. This option allows you to view the ASCII values of the objects in order to perform parameterization (enabled by default).
  - **Limit Object Size (bytes).** Limits serializable objects to the specified value. Objects whose size exceeds this value, will not be given ASCII representation in the script. The default value is 3072.
  - **Ignore Serialized Objects.** Lists the serialized objects not to be unfolded when encountered in the recorded script. Separate objects with commas.
  - **Serialization Delimiter.** Indicates the delimiter separating the elements in the ASCII representation of objects. VuGen will only parameterize strings contained within these delimiters. The default delimiter is '#'.
- **Unfold Arrays.** Expands array elements of serialized objects in ASCII representation. If you disable this option and an object contains an array, the object will not be expanded. By default, this option is enabled—all deserialized objects are totally unfolded.
  - **Limit Array Entries.** Instructs the recorder not to open arrays with more than the specified number of elements. The default value is 200.

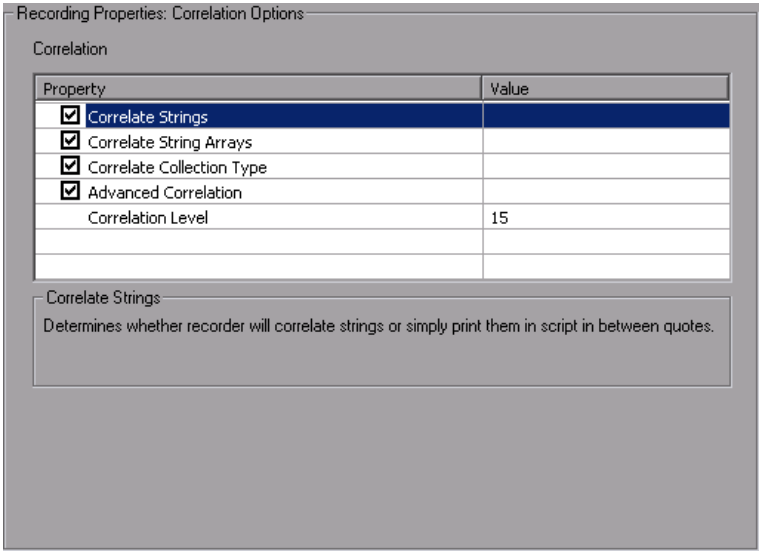
To set the Serialization options:

1. Click **Options** in the Start Recording dialog box and select the **Recording Properties:Serialization Options** node.
2. Set the options as desired. To set all options to their default values, click **Use Defaults**.
3. Click **OK** to close the dialog box and begin recording.

For more information on serialization, see "Using the Serialization Mechanism" on page 492.

Correlation Options

The **Correlation** options let you enable automatic correlation, and control its depth.



The following options are available:

- **Correlate Strings.** Correlate all strings that require correlation. If this option is disabled, VuGen prints them in the script, wrapped in quotes (disabled by default).
- **Correlate String Arrays.** Correlate text within string arrays (enabled by default).
- **Correlate Collection Type.** Correlates objects from the Collection class for JDK 1.2 and higher (disabled by default).
- **Advanced Correlation.** Enables deep correlation in CORBA container constructs and arrays (enabled by default).
- **Correlation Level.** Indicates the level of deep correlation, the number of inner containers to be scanned (15 by default).

To set the Correlation options:

1. Click **Options** in the Start Recording dialog box and select the **Recording Properties:Correlation Options** node.

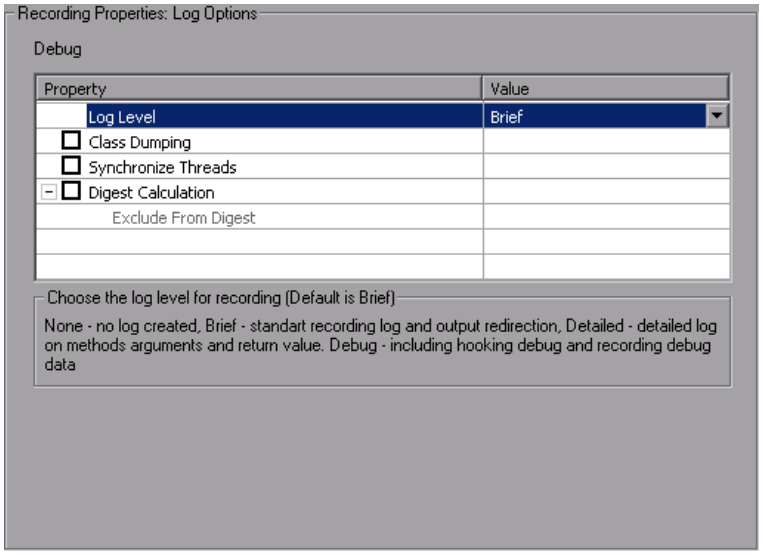
2. Enable the desired options, or for options that require values, enter the desired value. To set all options to their default values, click **Use Defaults**.

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

For more information about correlation, see Chapter 27, "Java - Correlating".

Log Options

The **Log** recording options let you determine the level of debug information generated during recording.



The following options are available:

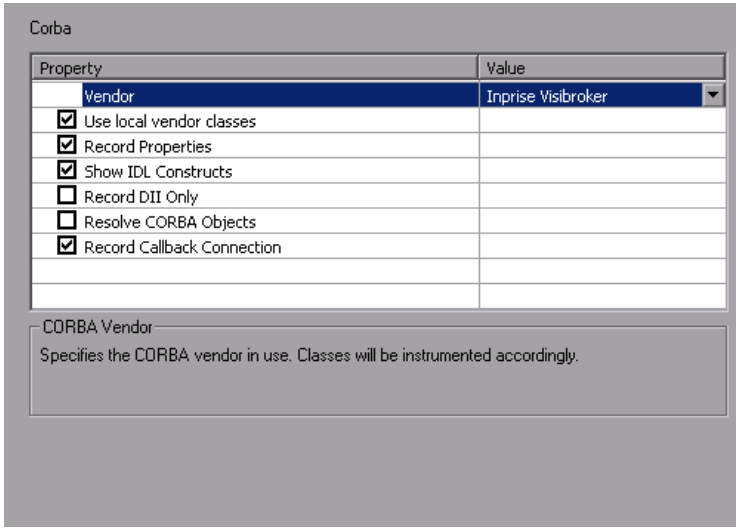
- **Log Level.** The level of recording log to generate.
  - **None**. No log file is created
  - **Brief**. Generates a standard recording log and output redirection
  - **Detailed**. Generates a detailed log for methods, arguments, and return values.
  - **Debug.** Records hooking and recording debug information, along with all of the above.
- **Class Dumping.** Dumps all of the loaded classes to the script directory. (disabled by default).
- **Synchronize Threads.** For multi-threaded applications, instructs VuGen to synchronize between the different threads (disabled by default).
- **Digest Calculation.** Generate a digest of all recorded objects (disabled by default).
  - **Exclude from Digest.** A list of objects not to be included in the digest calculation.

To set the Log options:

1. Click **Options** in the Start Recording dialog box and select the **Recording Properties:Log Options** node.
2. Select a Log level: None, Brief, Detailed, or Debug.
3. Enable the desired options, or for options that require values, enter the desired value.
4. To set all options to their default values, click **Use Defaults**.
5. Click **OK** to close the dialog box and begin recording.

CORBA Options

The following options are specific to CORBA-Java. These options let you set the CORBA specific recording properties and several callback options.



The following options are available:

- **Vendor.** The CORBA vendors **Inprise Visibroker**, **Iona OrbixWeb**, or **Bea Weblogic**.
- **Use local vendor classes.** Use local vendor classes and add the **srv** folder to the BOOT classpath. If you disable this option, VuGen uses network classes and adds the script's classes to the classpath (enabled by default).
- **Record Properties.** Instructs VuGen to record system and custom properties related to the protocol (enabled by default).
- **Show IDL Constructs.** Displays the IDL construct that is used when passed as a parameter to a CORBA invocation (enabled by default).
- **Record DLL only.** Instructs VuGen to record only on a DLL level (disabled by default).
- **Resolve CORBA Objects.** When correlation fails to resolve a CORBA object, recreate it using its binary data (disabled by default).

➤ **Record CallBack Connection.** Instructs VuGen to generate a connect statement for the connection to the ORB, for each callback object (disabled by default).

To set the CORBA recording options:

1. Click **Options** in the Start Recording dialog box and select the **Recording Properties:Corba Options** node.
2. Enable or disable the options as desired.
3. To set all options to their default values, click **Use Defaults**.
4. Click **OK** to close the dialog box and begin recording.

76

Setting Script Generation Preferences

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

This chapter includes:

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 113.

After you select a generation language, you can enable language-specific recording options which instruct the recorder what to include in the script and how to generate it.

If at least one of the protocols you are recording has multi-protocol capabilities, the *Script* 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 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 and control flow statements.

The following sections describe the scripting options. For all scripts, see "Applying the Basic Options" on page 1213. To set the correlation options for non-C scripts, see "Understanding the Correlation Options" on page 1215.

For further instructions, see "Setting Script Recording Options" on page 1216.

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 1216.

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 1216.

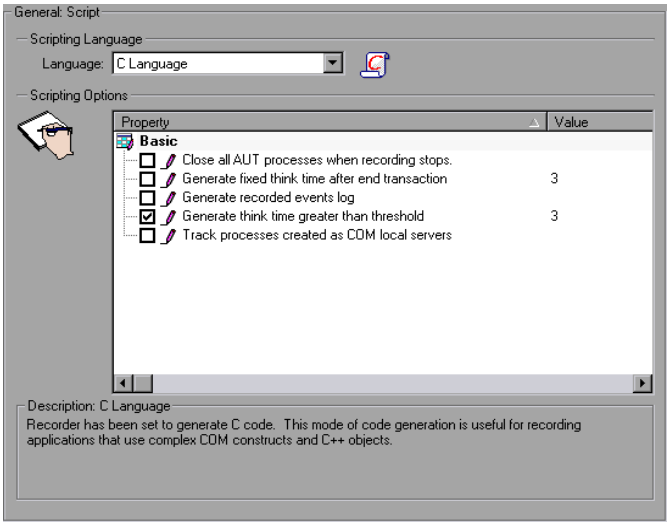
Setting Script Recording Options

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. Select **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.

77

Setting Recording Options for Web Vusers

Before recording a Web session, you can customize the recording options.

This chapter includes:

The following information applies to Web (HTTP/HTML), Web (Click and Script), Web/WinSocket, Oracle Web Applications 11i, and PeopleSoft Enterprise Vuser scripts.

About Setting Recording Options

VuGen enables you to generate Web Vuser scripts by recording typical processes that users perform on your Web site.

Before recording, you can configure the Recording Options and specify the information to record, the browser or client with which to record, and designate the content for your scripts.

You can set the common HTTP Properties recording options, such as proxy settings and other advanced settings. For more information see "Web, Wireless, and Oracle NCA Recording Options" on page 1168

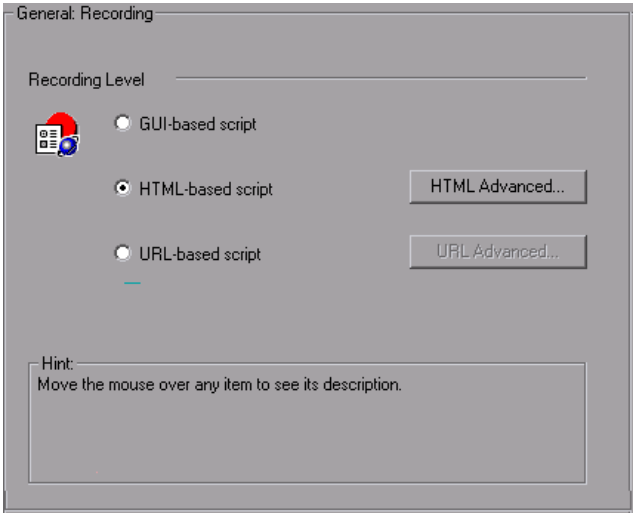
You can also set Correlation recording options for Web Vuser scripts. For more information, see Chapter 53, "Web (HTTP/HTML) Correlation Rules".

Understanding the Recording Levels

VuGen lets you specify what information to record and which functions to use when generating a Vuser script, by selecting a recording level. The recording level you select, depends on your needs and environment. The available levels are **GUI-based script, HTML-based script,** and **URL-based script**. For Web HTTP/HTML Vusers, only the two latter options are available.

Use the following guidelines to decide which recording level to use:

- For most applications, including those with JavaScript, use a **GUI-based script**. This level is also recommended for PeopleSoft Enterprise and Oracle Web Applications 11i Vusers.
- For browser applications with applets and VB Script, create an **HTML-based script**.
- For non-browser applications, use a **URL-based script**.



The **GUI-based script** option instructs VuGen to record HTML actions as context sensitive GUI functions such as **web_text_link**.

```
/* GUI-based mode - CS type functions with JavaScript support*//
vuser_init()
{
web_browser("WebTours",
        DESCRIPTION,
        ACTION,
        "Navigate=http://localhost:1080/WebTours/",
        LAST);

web_edit_field("username",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=text",
        "Name=username",
        "FrameName=navbar",
        ACTION,
        "SetValue=jojo",
        LAST);
…
```

The HTML-based script level generates a separate step for each HTML user action. The steps are also intuitive, but they do not reflect true emulation of the JavaScript code.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("WebTours",
        "URL=http://localhost/WebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

The **URL-based script** mode option instructs VuGen to record all browser requests and resources from the server that were sent due to the user's actions. It automatically records every HTTP resource as URL steps (**web_url** statements). For normal browser recordings, it is not recommended to use the URL-based mode since is more prone to correlation related issues. If, however, you are recording pages such as applets and non-browser applications, this mode is ideal.

URL-based scripts are not as intuitive as the HTML-based scripts, since all actions are recorded as web_url steps instead of web_link, web_image, and so on.

```
/* URL-based mode - only web_url functions */
…
web_url("spacer.gif",
        "URL=http://graphics.hplab.com/images/spacer.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=",
        "Mode=HTTP",
        LAST);

web_url("calendar_functions.js",
        "URL=http://www.im.hplab.com/travelp/calendar_functions.js",
        "Resource=1",
        "RecContentType=application/x-javascript",
        "Referer=",
        "Mode=HTTP",
        LAST);
…
```

You can switch recording levels and advanced recording options while recording, provided that you are not recording a multi-protocol script. The option of mixing recording levels is available for advanced users for performance testing.

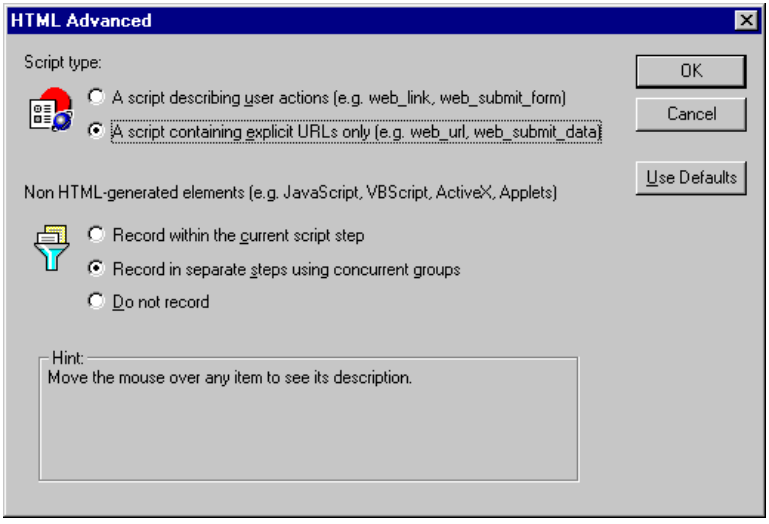
You can also regenerate a script after recording, using a different method than the original recording. For example, if your record a script on an HTML-based level, you can regenerate it on a URL-based level. To regenerate a script, select **Tools** > **Regenerate Script** and click **Options** to set the recording options for the regeneration.

Setting Advanced HTML-Based Options

The **HTML-based** option, which is the default recording level for Web (HTTP/HTML) Vusers, instructs VuGen to record HTML actions in the context of the current Web page. It does not record all resources during the recording session, but downloads them during replay.

VuGen lets you set advanced options for HTML-based script in the following areas:

- Specifying Script Types
- Handling Non HTML-Generated Elements



Specifying Script Types

For HTML-based scripts, you can specify the type of script:

- A script describing user actions
- A script containing explicit URLs only

The first option, **A script describing user actions**, is the default option. It generates functions that correspond directly to the action taken. It creates URL (**web_url**), link (**web_link**), image (**web_image**), and form submission (**web_submit_form**) functions. The resulting script is very intuitive and resembles a context sensitive recording.

```
/* HTML-based mode - a script describing user actions*//
...
web_url("WebTours",
        "URL=http://localhost/WebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

The second option, **A script containing explicit URLs only**, records all links, images and URLs as **web_url** statements, or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions. The resulting script is less intuitive. This mode is useful for instances where many links within your site have the same link text. If you record the site using the first option, it records an ordinal (instance) for the link, but if you record using the second option, each link is listed by its URL. This facilitates parameterization and correlation for that step.

The following segment illustrates a session recorded with a script containing explicit URLs only selected:

```
/* A HTML-based script containing explicit URLs only*//
…
web_url("Click Here For Additional Restrictions",
        "URL=http://www.hplab.com/restrictions.html",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.hplab.com/home?…
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

web_url("buttonhelp.gif",
        "URL=http://www.hplab.com/home?com/rstr?BV_EngineID...,
        "TargetFrame=main",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.hplab.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
…
```

Handling Non HTML-Generated Elements

Many Web pages contain non-HTML elements, such as applets, XML, ActiveX elements, or JavaScript. These non-HTML elements usually contain or retrieve their own resources. For example, a JavaScript **js** file, called from the recorded Web page, may load several images. An applet may load an external text file. Using the following options, you can control how VuGen records non HTML-generated elements.

The following options are available:

- Record within the current script step (default)
- Record in separate steps using concurrent groups
- Do not record

The first option, **Record within the current script step**, does not generate a new function for each of the non HTML-generated resources. It lists all resources as arguments of the relevant functions, such as web_url, web_link, and web_submit_data. The resources, arguments of the Web functions, are indicated by the EXTRARES flag. In the following example, the web_url function lists all of the non HTML-generated resources loaded on the page:

```
web_url("index.asp",
    "URL=http://www.daisy.com/index.asp",
    "TargetFrame=",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t2.inf",
    "Mode=HTML",
    EXTRARES,
    "Url=http://www.daisy.com/ScrollApplet.class", "Referer=", ENDITEM,
    "Url=http://www.daisy.com/board.txt", "Referer=", ENDITEM,
    "Url=http://www.daisy.com/nav_login1.gif", ENDITEM,
    …
    LAST);
```

The second option, **Record in separate steps using concurrent groups**, creates a new function for each one of the non HTML-generated resources—it does not include them as items in the page's functions (such as **web_url**, **web_link**, and so on). All of the **web_url** functions generated for a resource, are placed in a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**).

In the following example, the above session was recorded with this option selected. A web_url function was generated for the applet and text file loaded with the applet:

```
web_url("index.asp",
        "URL=http://www.daisy.com/index.asp",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t2.inf",
        "Mode=HTML",
        LAST);

web_concurrent_start(NULL);
    web_url("ScrollApplet.class",
        "URL=http://www.daisy.com/ScrollApplet.class",
        "Resource=1",
        "RecContentType=application/octet-stream",
        "Referer=",
        LAST);

    web_url("board.txt",
        "URL=http://www.daisy.com/board.txt",
        "Resource=1",
        "RecContentType=text/plain",
        "Referer=",
        LAST);
web_concurrent_end(NULL);
```

The third option, **Do not record**, instructs VuGen not to record any of the resources generated by non-HTML elements.

Note that when you work in HTML-Based mode, VuGen inserts the TargetFrame attribute in the **web_url** statement. VuGen uses this information to display the Web page correctly in the run-time browser and Test Result report.

```
web_url("buttonhelp.gif",
        "URL=http://www.hplab.com/home?com/rstr?BV_EngineID...,
        "TargetFrame=main",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.hplab.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
```

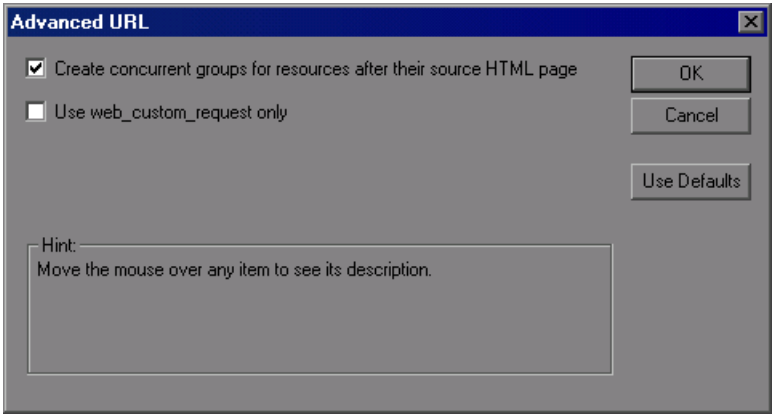
When you record the URL-based mode, VuGen records the content of all frames on the page and therefore omits the **TargetFrame** attribute.

Setting Advanced URL-Based Options

The **URL-based** mode option instructs VuGen to record all requests and resources from the server. It automatically records every HTTP resource as URL steps (**web_url** statements), or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions, nor does it record frames.

VuGen lets you set advanced options for the URL recording mode in the following area:

- Resource Handling
- Generating Custom HTTP Requests



Resource Handling

In URL-based recording, VuGen captures all resources downloaded as a result of a browser request. By default, this option is enabled and VuGen records the resources in a concurrent group (enclosed by **web_concurrent_start** and **web_concurrent_end** statements) after the URL. Resources include files such as images, and **js** files. If you disable this option, the resources are listed as separate **web_url** steps, but not marked as a concurrent group.

The following segment illustrates a session recorded with the Create concurrent groups for resources after their source HTML page option enabled.

```
web_concurrent_start (NULL);
…
web_url("Click Here For Additional Restrictions",
        "URL=http://www.hplab.com/restrictions.html",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.hplab.com/home?…
        "Snapshot=t4.inf",
        "Mode=HTTP",
        LAST);

web_url("buttonhelp.gif",
        "URL=http://www.hplab.com/home?com/rstr?BV_EngineID...,
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.hplab.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTTP",
        LAST);
…
web_concurrent_end (NULL);
```

Note that the script includes **gif**, and **js** files. This mode also includes other graphic files and imported file such as **imp**, **txt**, and cascading style sheet (**css**) files.

Generating Custom HTTP Requests

When recording non-browser applications, you can instruct VuGen to record all HTTP requests as custom requests. VuGen generates a **web_custom_request** function for all requests, regardless of their content:

```
web_custom_request("www.hplab.com",
        "URL=http://www.hplab.com/",
        "Method=GET",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTTP",
        LAST);
```

Enabling EUC-Encoded Web Pages

(This option is only for Japanese Windows.) When working with non-Windows standard character sets, you may need to perform a code conversion. A character set is a mapping from a set of characters to a set of integers. This mapping forms a unique character-integer combination, for a given alphabet. Extended UNIX Code (EUC) and Shift Japan Industry Standard (SJIS) are non-Windows standard character sets used to display Japanese writings on Web sites.

Windows uses SJIS encoding, while UNIX uses EUC encoding. When a Web server is on a UNIX machine and the client is Windows, the characters in a Web site are not displayed on the client machine properly due to the difference in the encoding methods. This affects the display of EUC-encoded Japanese characters in a Vuser script.

During recording, VuGen detects the encoding of a Web page through its HTTP header. If the information on the character set is not present in the HTTP header, it checks the HTML meta tag. If the page does not send the character set information to the HTTP header or meta tag, VuGen does not detect the EUC encoding.

If you know in advance that a Web page is encoded in EUC, you can instruct VuGen to use the correct encoding during record. To record a page in EUC-encoding, enable the **EUC** option in the Recording Options **Recording** tab (only visible for Japanese Windows).

Enabling the **EUC** option, forces VuGen to record a Web page in EUC encoding, even when it is not EUC-encoded. You should, therefore, only enable this option when VuGen cannot detect the encoding from the HTTP header or the HTML meta tag, and when you know in advance that the page is EUC-encoded.

During recording, VuGen receives an EUC-encoded string from the Web server and converts it to SJIS. The SJIS string is saved in the script's Action function. However, for replay to succeed, the string has to be converted back to EUC before being sent back to the Web server. Therefore, VuGen adds a **web_sjis_to_euc_param** function before the Action function, which converts the SJIS string back to EUC.

In the following example, the user goes to an EUC-encoded Web page and clicks a link. VuGen records the Action function and adds the web_sjis_to_euc_param function to the script before the Action function.

```
web_sjis_to_euc_param("param_link","Search");
web_link("LinkStep","Text={param_link}");
```

Setting the Recording Level

This section describes the procedure for setting the recording levels and their advanced options.

To set the recording options:

1. Select **Tools** > **Recording Options** to open the Recording Options.
2. Select the **General:Recording** node in the Recording Options tree.
3. Select a recording mode: **GUI-based** (when available), **HTML-based**, or **URL-based.**
4. For GUI-based recording, open the recording options (Ctrl + F7) and select the **GUI Properties:Advanced** node to set additional options for capturing events during recording.

   For more information, see "Setting Advanced GUI Properties" on page 1181.

5 For HTML-based recording, click **HTML Advanced** to set additional options for script types and the handling of non-HTML elements.

Select a script type.

Select a method for handling non-HTML resources. For more information, see "Setting Advanced HTML-Based Options" on page 1222.

6 For URL-based recording, click **URL Advanced** to set additional script options for resource handling and cache enabling.

Select **Create concurrent groups for resources after their source HTML page** to enable the recording of resources and marking them as a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**).

Select **Enable cache** to use the browser cache during recording. If you enable this option, clear the **Clear cache before recording** check box to instruct VuGen not to clear the cache and use previously accessed pages.

Select **Use web_custom_request only** to generate all HTTP requests as **web_custom_request** functions. For more information about these options, see "Setting Advanced URL-Based Options" on page 1227.

7 For users of Japanese Windows, select the **EUC** option to instruct VuGen to use EUC encoding.

If you are recording a Web site whose pages use only the EUC Encoding (Japanese content), select the **EUC** option. VuGen converts the EUC string to SJIS and adds a **web_sjis_to_euc_param** function. If the server sends this information to the browser (in an HTTP header or an HTML Meta tag), you do not need to enable this option.

78

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.

This chapter includes:

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, select FTP.
- **New Entry.** Opens the Server Entry dialog box, allowing you to add a new mapping. See "Adding a New Server Entry" on page 1237.
- **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 1239.

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.

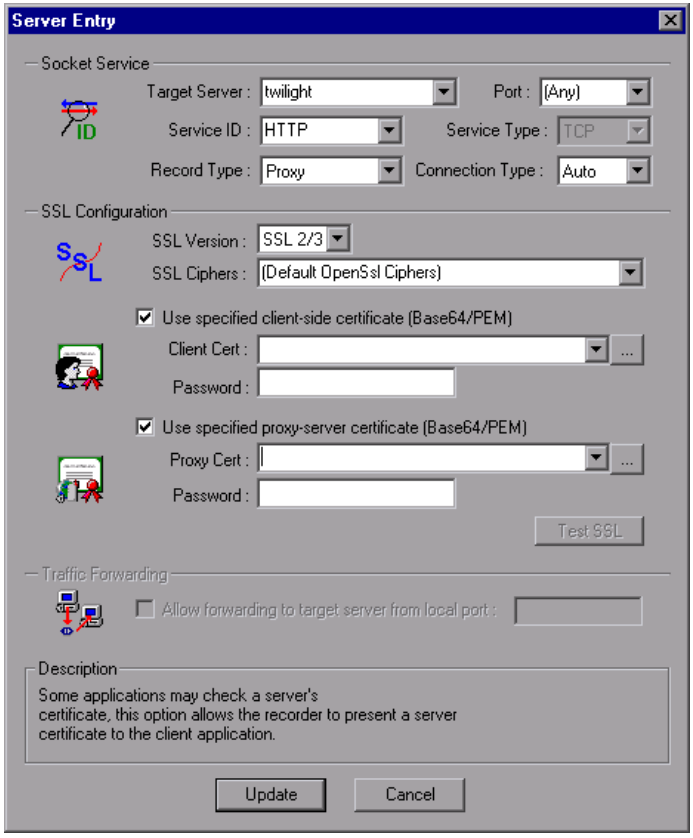
➤ **Forced mapping.** If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol to record that communication, even though 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 1242.

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 1242.

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, we recommend that you allow VuGen to use auto-detection to determine the protocol type. In most cases, VuGen's recorder is able to recognize the signatures of these protocols. It then automatically processes them according to the protocol specifications. In certain instances, however, VuGen may be unable to recognize the protocol. For example:

- The protocol signature closely resembles an existing protocol, resulting in erroneous processing.
- There is no unique signature for the protocol.
- The protocol uses SSL encryption, and therefore cannot be recognized on a WinSock level.

In all of the above cases, you can supply information to uniquely identify the server and port hosting the protocol.

For further instructions, see "Setting the Port Mapping Recording Options" on page 1242.

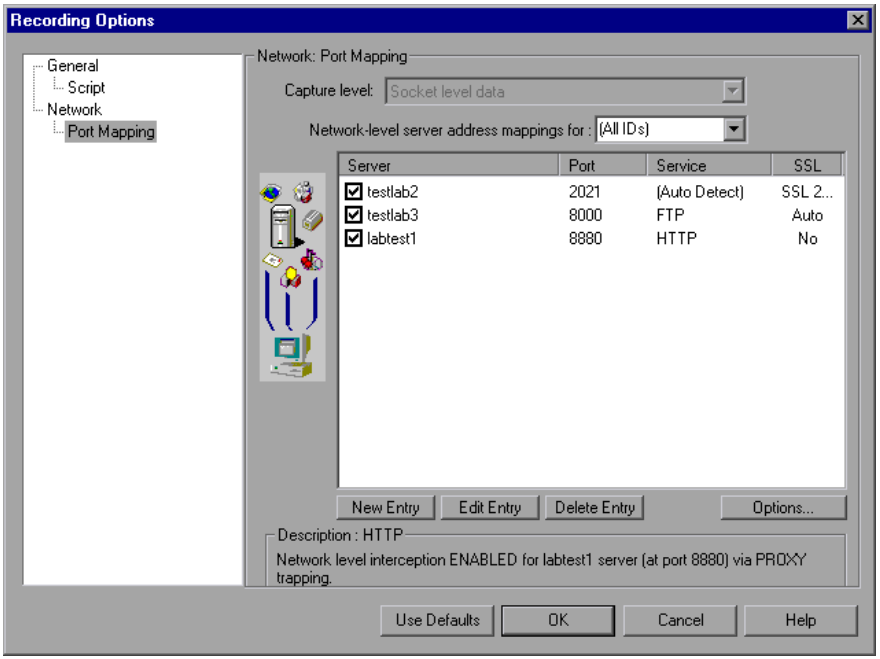
Setting the Port Mapping Recording Options

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

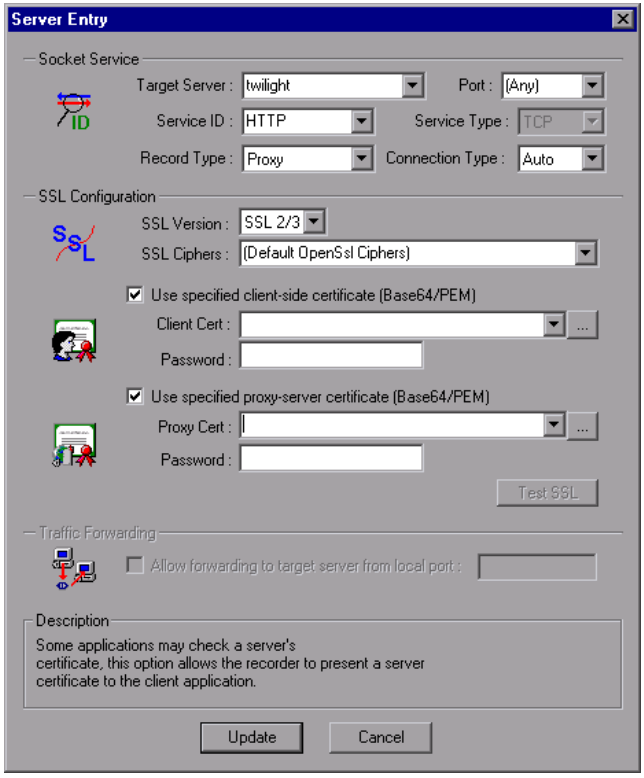
- The toolbar button: 
- The keyboard shortcut: Ctrl+F7
- The Tools menu: select **Tools** > **Recording Options**

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.

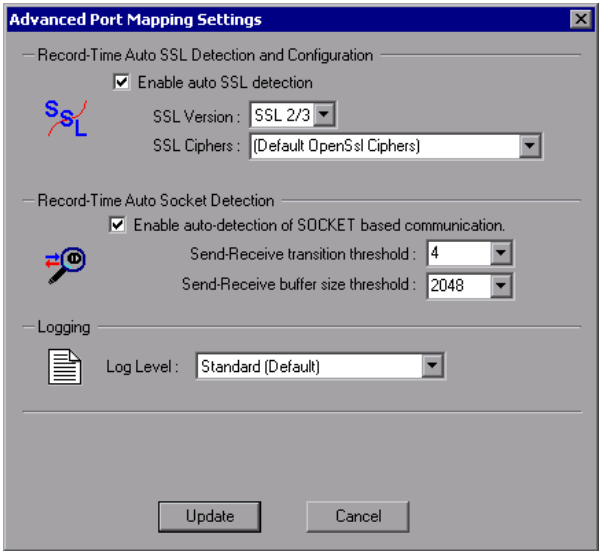
Specify the **SSL Version**, **SSL Cipher**. To use a certificate, select **Use specified client-side certificate** or **Use specified proxy-server certificate** and specify the user information.

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**.

Part 5

Run-Time Settings

79

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.

This chapter includes:

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**.
- Select **Vuser** > **Run-Time Settings**.

You can also modify the run-time settings from the LoadRunner Controller. For more information, see 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.

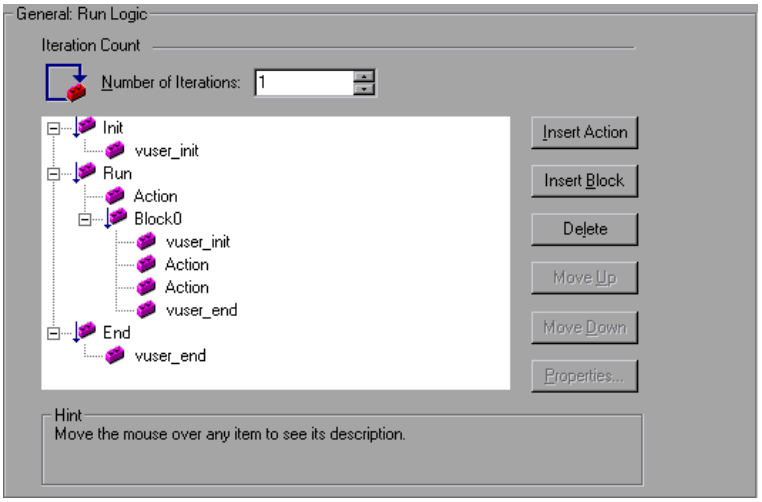
Configuring Run Logic Run-Time Settings (multi-action)

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 1258.

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.

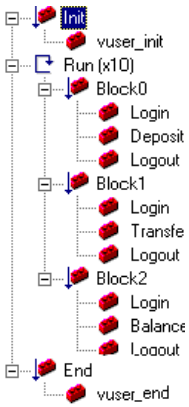
➤ **Sequence.** You can set the order of actions within your script. You can also indicate whether to perform actions sequentially or randomly.

- **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.

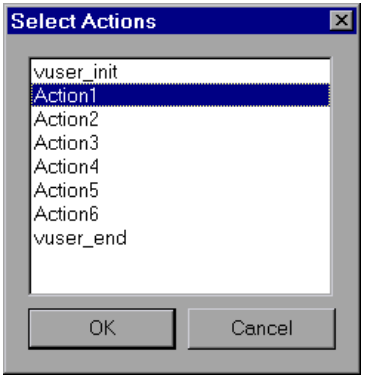
In the following example, *Block0* performs a deposit, *Block1* performs a transfer, and *Block2* submits a balance request. The *Login* and *Logout* actions are common to the three blocks.



You configure each block independently—its sequence and iterations.

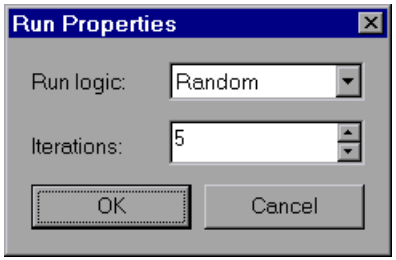
To configure actions and action blocks:

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.

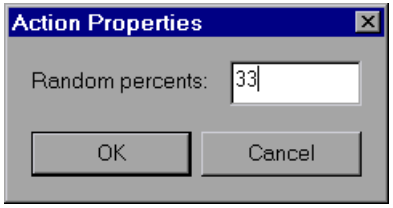


10 Select *Sequential* or *Random* from the **Run Logic** list, indicating to VuGen whether to run the actions sequentially or randomly.

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 select **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 1258.

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.

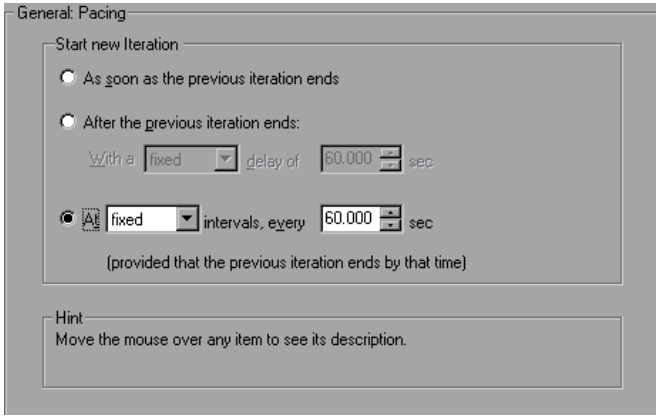
For further instructions about setting the Pacing options, see "Configuring Pacing Run-Time Settings (multi-action)" on page 1257.

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**.

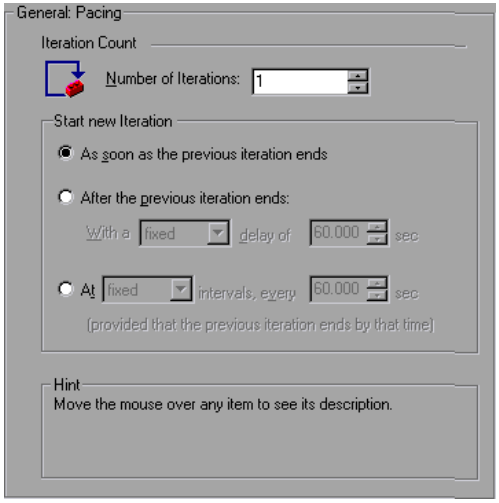
Setting Pacing and Run Logic Options (single action)

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 1256.

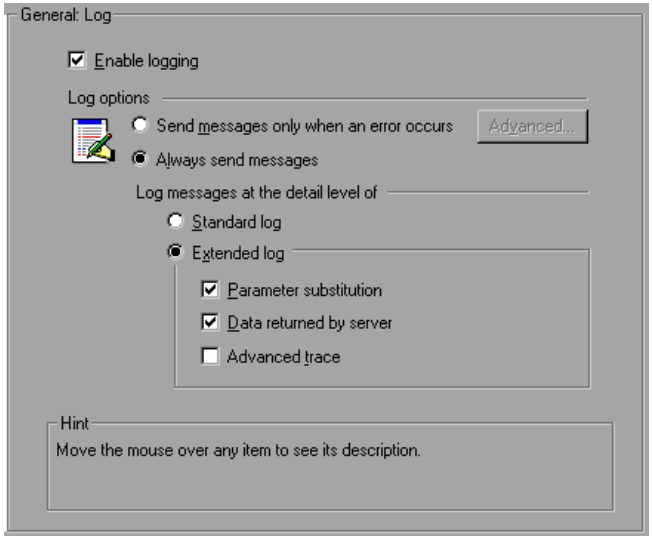
Configuring the Log Run-Time Settings

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 1263.

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 70, "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, see 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

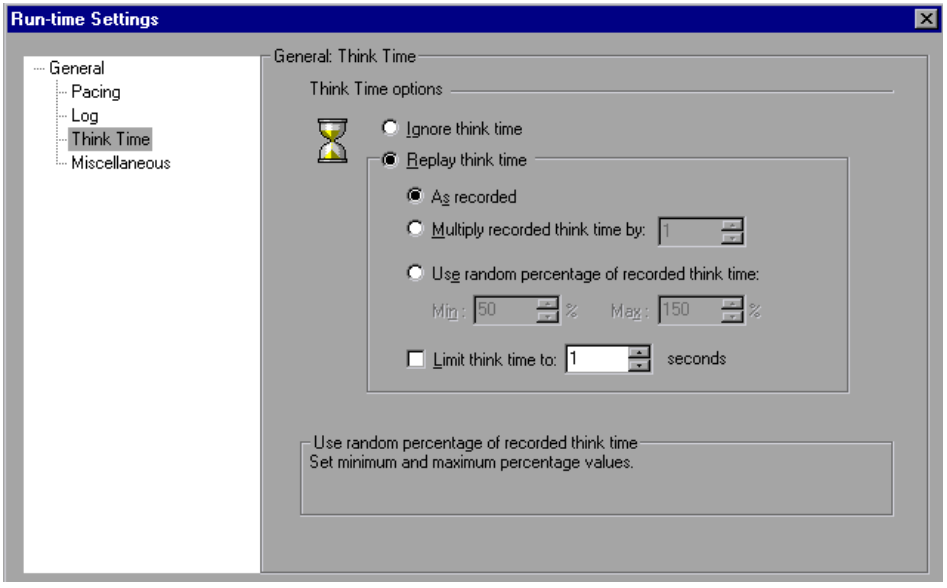
Vuser *think time* emulates the time that a real user waits between actions. For example, when a user receives data from a server, the user may wait several seconds to review the data before responding. This delay is known as the *think time*. VuGen uses **lr_think_time** functions to record think time 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, see 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.

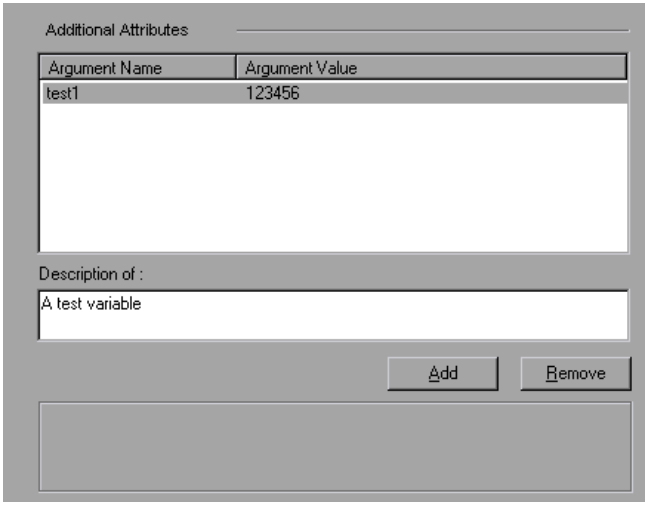
Configuring Additional Attributes Run-Time Settings

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 6, "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.

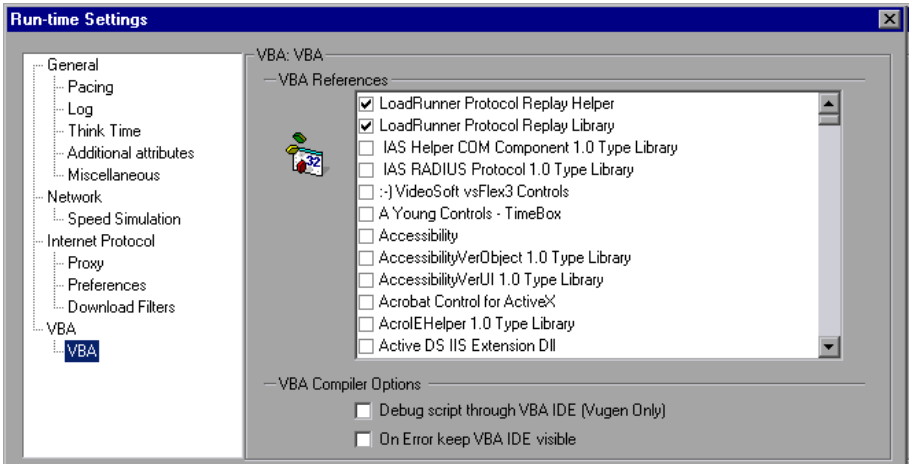
Setting the VB Run-Time Settings

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 Select **OK** to apply the run-time settings.

80

Configuring Network Run-Time Settings

To simulate the speed over a network, you configure the Network run-time settings.

This chapter includes:

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 79, "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:

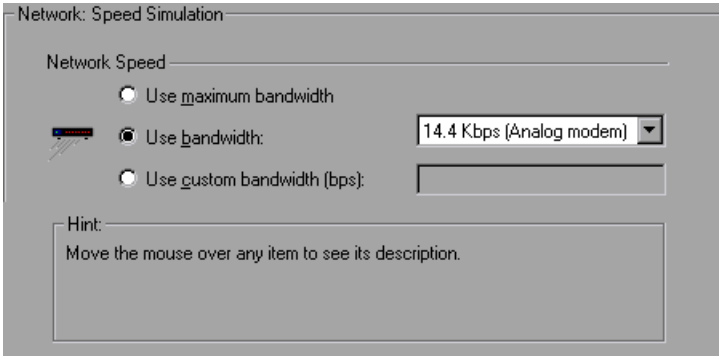
To display the Run-Time Settings dialog box:

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

Note that you can also modify the run-time settings from the LoadRunner Controller. For more information, see 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.



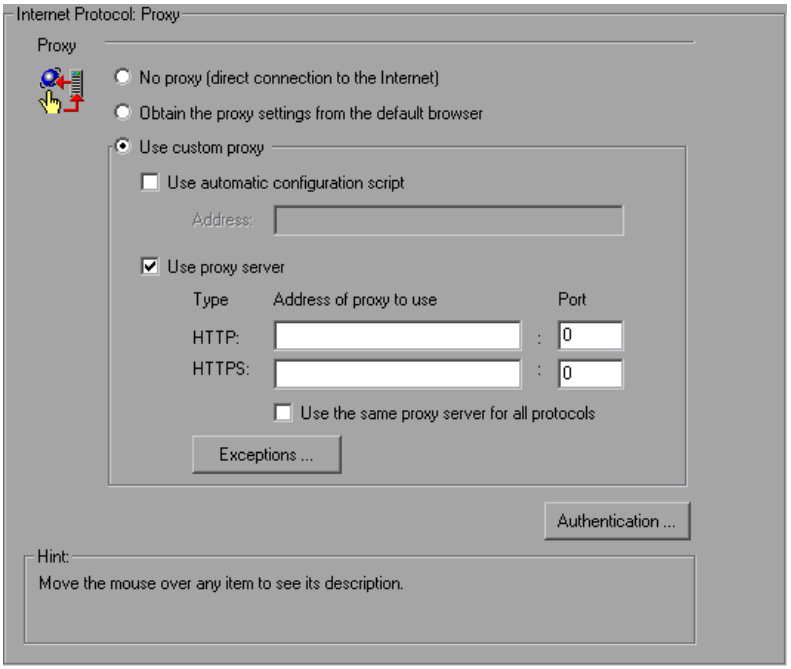
Speed Simulation

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.

Setting Proxy Options

You use the **Internet Protocol:Proxy** node of the Run-Time Settings tree, to set the proxy-related settings.



The following proxy options are available in the Run-Time settings.

- **No proxy.** All Vusers should use direct connections to the Internet. This means that the connection is made without using a proxy server.
- **Obtain the proxy settings from the default browser.** All Vusers use the proxy settings of the default browser from the machine upon which they are running.

➤ **Use custom proxy.** All Vusers use a custom proxy server. You can supply the actual proxy server details or the path of a proxy automatic configuration script (.**pac** file) that enables automatic configuration. (See "Setting the Automatic Proxy Configuration" on page 1280.)

To supply the details of the server, you specify its IP address or name and port. You can specify one proxy server for all HTTP sites, and another proxy server for all HTTPS (secure) sites.

After providing the proxy information, you can specify Authentication information for the proxy server, and indicate Exceptions to the proxy rules.

Note: To instruct the Vusers to wait for the proxy response during replay, and not to assume that the proxy supports basic authentication, add the following statement:
web_set_sockets_option("PROXY_INITIAL_BASIC_AUTH", "0");

Authentication

If the proxy server requires authentication for each Vuser, use this dialog box to enter the relevant password and user name.

➤ **User Name.** Enter the user name that Vusers will use to access the proxy server.

➤ **Password.** Enter the password required by Vusers to access the proxy server.

Note: To add authentication dynamically during recording, or to add authentication for multiple proxy servers, use the **web_set_user** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

Exceptions

You can specify that all Vusers use a specified proxy server. In such a case, if there are any URLs that you want Vusers to access directly, that is, without using the proxy server, enter the list of these URLs in the text box.

- **Do not use proxy server for addresses beginning with.** Enter the addresses you want to exclude from the proxy server. Use semicolons to separate entries.
- **Do not use proxy server for local (intranet) addresses.** Select this check box to exclude local addresses, such as those from an Intranet, from the proxy server.

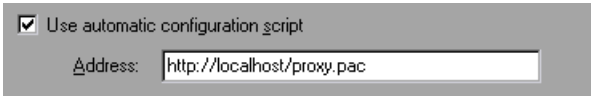
Setting the Automatic Proxy Configuration

Automatic Proxy Configuration is a feature supported by most browsers. This feature allows you to specify a JavaScript file (usually with a **.pac** extension) containing proxy assignment information. This script tells the browser when to access a proxy server and when to connect directly to the site, depending on the URL. In addition, it can instruct the browser to use a specific proxy server for certain addresses and another server for other addresses.

You can instruct VuGen or your Internet Explorer browser to work with a configuration script. You specify a file for the automatic proxy configuration, so that when the Vuser runs the test, it uses the rules from the proxy file.

To specify a configuration script in VuGen:

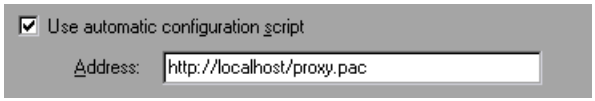
1. Select **Vuser** > **Run-TIme Settings**, and select the **Internet Protocol:Proxy** node.
2. Select **Use custom proxy** and select the **Use automatic configuration script** option. Specify the location of the script.



To specify a configuration script in Internet Explorer (IE):

1. Select **Tools** > **Internet Options**, and select the Connections tab.
2. Click the **LAN Settings** button. The LAN Settings dialog box opens.

3 Select the **Use automatic configuration script** option, and specify the location of the script.



To track the behavior of the Vusers, generate a log during text execution and view the Execution Log tab or the mdrv.log file. The log shows the proxy servers that were used for each URL. In the following example, VuGen used a direct connection for the URL australia.com, but the proxy server aqua, for the URL http://www.google.com.

```
Action1.c(6): t=1141ms: FindProxyForURL returned DIRECT
Action1.c(6): t=1141ms: Resolving australia.com
Action1.c(6): t=1141ms: Connecting to host 199.203.78.255:80
…
Action1.c(6): t=1281ms: Request done "http://australia.com/GetElementByName.htm"

…
Action1.c(6): web_url was successful, 357 body bytes, 226 header bytes
Action1.c(15): web_add_cookie was successful
Action1.c(17): t=1391ms: FindProxyForURL returned PROXY aqua:2080
Action1.c(17): t=1391ms: Auto-proxy configuration selected proxy aqua:2080
Action1.c(17): t=1391ms: Resolving aqua
Action1.c(17): t=1391ms: Connecting to host 199.203.139.139:2080
…
Action1.c(17): t=1578ms: 168-byte request headers for "http://www.google.com/"
(RelFrameId=1)
Action1.c(17):     GET http://www.google.com/ HTTP/1.1\r\n
```

Setting Proxy Run-Time Settings

The following section discusses the steps required for configuring the proxy Run-Time settings.

To set the proxy settings:

1 Open the Run-Time settings. Click the **Run-Time Settings** button on the VuGen toolbar or select **Vuser** > **Run-Time Settings**.

2 Click the **Internet Protocol:Proxy** node.

3 Select the desired proxy option: **No proxy, Obtain the proxy settings from the default browser, or Use custom proxy.**

4 If you specified a custom proxy:

- ➤ indicate the IP addresses for the HTTP and HTTPS proxy servers
- ➤ To use a **pac** or JavaScript file to indicate the proxy, select the **Use automatic configuration script** option and specify the script location. You can specify either a web location beginning with http:// (for example, **http://hostname/proxy.pac**), or a location on the file server, for example, **C:\temp\proxy.pac**.

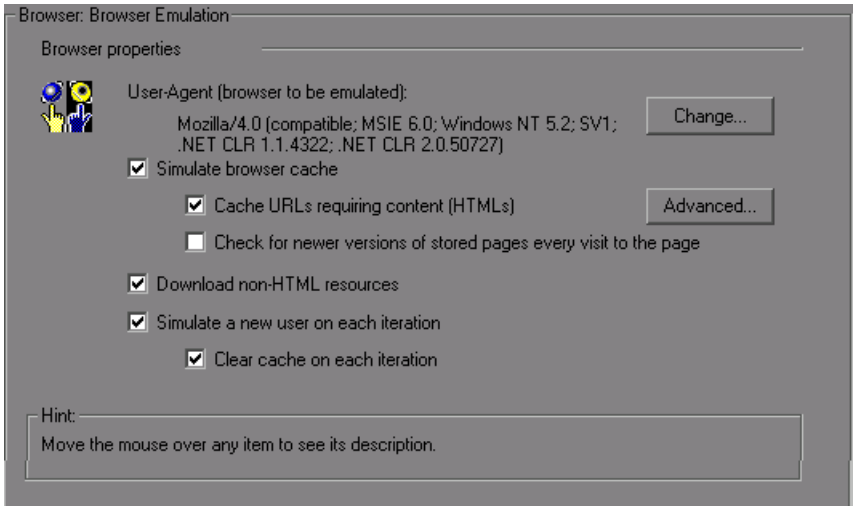
5 To specify URLs that you want Vusers to access directly, without the proxy server, click **Exceptions** and then supply the list of these URLs. In the Exceptions dialog box, you can also specify direct access to local (intranet) addresses.

6 If the proxy server requires authentication, click **Authentication**, and then supply the relevant password and user name.

7 Select the **Use the same proxy server for all protocols** check box to instruct the Vusers to use the same proxy server for all Internet protocols (HTTP, HTTPS) rather than specifying a specific server for secure sites.

Setting Browser Emulation Properties

You use the **Browser:Browser Emulation** node in the Run-Time Settings tree to set the browser properties of your testing environment.



Browser Properties

You can set the browser properties in the following areas:

- User-Agent (browser to be emulated)
- Simulate browser cache
- Download non-HTML resources
- Simulate a new user each iteration

You can also set advanced options for caching and checking for newer resources.

User-Agent (browser to be emulated)

Whenever a Vuser sends a request to a Web server, the request includes an HTTP header. The first line of text contains a verb (usually "GET" or "POST"), the resource name (for example "pclt/default.htm"), and the version of the protocol (for example "HTTP/1.0"). Subsequent lines contain "header information" in the form of an attribute name, a colon, and some value. The request ends with a blank line.

All Internet Vuser headers include a **User-Agent** header that identifies the type of browser (or toolkit for Wireless) that is being emulated. For example,

```
User-Agent: Mozilla/3.01Gold (WinNT; I)
```

identifies the Browser as Netscape Navigator Gold version 3.01 running under Windows NT on an Intel machine.

Click **Change** from the Browser emulation node, to specify the browser information to include in the header. You can specify that a Web Vuser emulate any of the standard browsers. Alternatively, for non-browser HTTP applications, you can specify the HTTP client to match a specific user's application. In this case, you must supply a **Custom User Agent** string that is included in all subsequent HTTP headers. By default, the user-agent emulates the Microsoft Internet Explorer 5.5 browser agent.

Simulate browser cache

This option instructs the Vuser to simulate a browser with a cache. A cache is used to keep local copies of frequently accessed documents and thereby reduces the time connected to the network. By default, cache simulation is enabled. When the cache is disabled, Vusers will ignore all caching functionality and download all of the resources for every request.

Note that even if you disable the cache simulation, each resource is only downloaded once for each page, even if it appears multiple times. A resource can be an image, a frame, or another type of script file.

When running multiple Vusers as in LoadRunner and Performance Center, every Vuser uses its own cache and retrieves images from the cache. If you disable this option, all Vusers emulate a browser with no cache available.

You can modify your Run-Time settings to match your browser settings for Internet Explorer, as follows:

Browser Setting	Run-Time Setting
Every visit to the page	Select **Simulate Browser Cache** and enable **Check for newer versions of stored pages every visit to the page**.
Every time you start Internet Explorer	Select **Simulate Browser Cache** only
Automatically	Select **Simulate Browser Cache** only
Never	Select **Simulate Browser Cache** and disable **Check for newer versions of stored pages every visit to the page**.

You can also set the following two browser cache options:

➤ **Cache URLs requiring content (HTML).** This option instructs VuGen to cache only the URLs that require the HTML content. The content may be necessary for parsing, verification, or correlation. When you select this option, HTML content is automatically cached. This option is enabled by default.

Tip: To decrease the memory footprint of the virtual users, disable this option, unless it is an explicit requirement for your test.

To add more content types to the list of cached types, click **Advanced**. For more information, see "Cache URLs Requiring Content - Advanced" on page 1287. Note that if you enable the parent option **Simulate browser cache**, but disable this option, VuGen nevertheless stores the graphic files.

➤ **Check for newer versions of stored pages every visit to the page.** This setting instructs the browser to check for later versions of the specified URL, than those stored in the cache. When you enable this option, VuGen adds the "If-modified-since" attribute to the HTTP header. This option brings up the most recent version of the page, but also generates more traffic during the scenario or session execution. By default, browsers do not check for newer resources, and therefore this option is disabled. Configure this option to match the settings in the browser that you want to emulate.

Download non-HTML resources

Instructs Vusers to load graphic images when accessing a Web page during replay. This includes both graphic images that were recorded with the page, and those which were not explicitly recorded along with the page. When real users access a Web page, they wait for the images to load. Therefore, enable this option if you are trying to test the entire system, including end-user time (enabled by default). To increase performance and not emulate real users, disable this option.

Tip: Disable this option if you experience discrepancies in image checks, since some images vary each time you access a Web page (for example, advertiser banners).

Simulate a new user each iteration

Instructs VuGen to reset all HTTP contexts between iterations to their states at the end of the **init** section. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. It deletes all cookies, closes all TCP connections (including keep-alive), clears the emulated browser's cache, resets the HTML frame hierarchy (frame numbering will begin from 1) and clears the user-names and passwords. This option is enabled by default.

➤ **Clear cache on each iteration.** Clears the browser cache for each iteration in order to simulate a user visiting a Web page for the first time. Clear the check box to disable this option and allow Vusers to use the information stored in the browser's cache, simulating a user who recently visited the page.

Cache URLs Requiring Content - Advanced

The Advanced dialog box lets you specify the URL content types that you want to store in the cache. This dialog box is accessible from the Run-time Settings - **Browser:Browser Emulation** node.

Note that changes to the advanced settings for multiple groups simultaneously, are not supported—edit each group's settings individually.

To add a content type:

1. Enable the **Specify URLs requiring content in addition to HTML page** option.
2. Click the plus sign to add additional content types, such as text/plain, text/xml, image/jpeg, and image/gif. Enter the content name in the text box.
3. To remove a content type from the list, select it and click the minus sign.

Setting Internet Preferences

You use the **Internet Protocol:Preferences** node in the Run-Time Settings tree, to set the settings related to the following areas:

- Image and Text Checks
- Generating Web Performance Graphs
- Advanced Web Run-Time Options

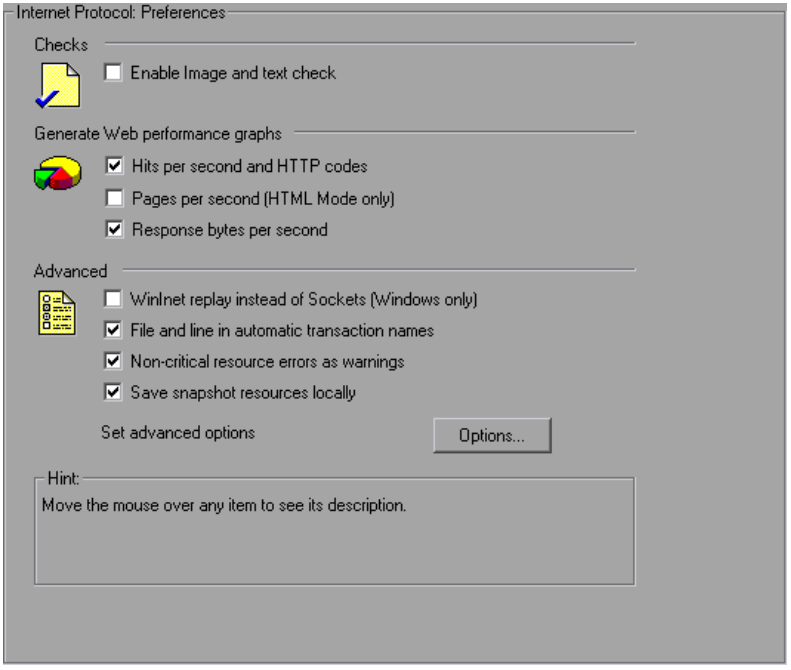


Image and Text Checks

The **Enable image and text checks** option allows the Vuser to perform verification checks during replay by executing the verification functions: **web_find** or **web_image_check**. This option only applies to statements recorded in HTML-based mode. Vusers running with verification checks use more memory than Vusers who do not perform checks (disabled by default).

Generating Web Performance Graphs

Instructs a Vuser to collect data used to create Web Performance graphs. You view the **Hits per Second**, **Pages per Second,** and **Response Bytes per Second (Throughput)** graphs during test execution using the online monitors and after test execution using the Analysis. You view the Component Breakdown graph after test execution using the Analysis. Select the types of graph data for the Vuser to collect.

Note: If you do not use the Web performance graphs, disable these options to save memory.

Advanced Web Run-Time Options

➤ **WinInet Replay.** Instructs VuGen to use the WinInet replay engine instead of the standard Sockets replay. VuGen has two HTTP replay engines: Sockets-based (default) or WinInet based. The WinInet is the engine used by Internet Explorer and it supports all of the features incorporated into the IE browser. The limitations of the WinInet replay engine are that it is not scalable, nor does it support UNIX. In addition, when working with threads, the WinInet engine does not accurately emulate the modem speed and number of connections.

VuGen's proprietary sockets-based replay is a lighter engine that is scalable for load testing. It is also accurate when working with threads. The limitation of the sockets-based engine is that it does not support SOCKS proxy. If you are recording in that type of environment, use the WinInet replay engine.

➤ **File and line in automatic transaction names.** Creates unique transaction names for automatic transactions by adding file name and line number to the transaction name (enabled by default).

Note: This option places additional information in the log file, and therefore requires more memory.

- **Non-critical item errors as warnings.** This option returns a warning status for a function which failed on an item that is not critical for load testing, such as an image or Java applet that failed to download. This option is enabled by default. If you want a certain warning to be considered an error and fail your test, you can disable this option. You can set a content-type to be critical by adding it to the list of Non-Resources. For more information, see "Specifying Non-Resource Content Types" on page 1174.
- **Save snapshot resources locally.** Instructs VuGen to save the snapshot resources to files on the local machine. This feature lets the Run-Time viewer create snapshots more accurately and display them quicker.

Additional Options for Internet Preferences

Click the **Options** button in the Advanced section of the Preferences node to set advanced options in the following areas: DNS caching, HTTP version, Keep-Alive HTTP connections, Accept server-side compression, Accept-Language headers, HTTP-request connect timeout, HTTP-request receive timeout, Network buffer size, and Step download timeout.

HTTP

- **HTTP version.** Specifies which version HTTP to use: version 1.0 or 1.1. This information is included in the HTTP request header whenever a Vuser sends a request to a Web server. The default is HTTP 1.1. HTTP 1.1 supports the following features:
  - Persistent Connections—see "Keep-Alive HTTP connections" below.
  - HTML compression—see "Performing HTML Compression" on page 1299.
  - Virtual Hosting—multiple domain names sharing the same IP address.
- **Keep-Alive HTTP connections.** Keep-alive is a term used for an HTTP extension that allows persistent or continuous connections. These long-lived HTTP sessions allow multiple requests to be sent over the same TCP connection. This improves the performance of the Web server and clients.

The keep-alive option works only with Web servers that support keep-alive connections. This setting specifies that all Vusers that run the Vuser script have keep-alive HTTP connections enabled (enabled by default).

- **Accept-Language request header.** Provides a comma-separated list of accepted languages. For example, **en-us**, **fr**, and so forth.
- **HTTP errors as warnings.** Issues a warning instead of an error upon failing to download resources due to an HTTP error.
- **HTTP-request connect timeout (seconds).** The time, in seconds, that a Vuser will wait for the connection of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user (default value is 120 seconds). Note that this timeout also applies to the time the Vuser will wait for a WAP connection, initiated by the **wap_connect** function.
- **HTTP-request receive timeout (seconds).** The time, in seconds, that a Vuser will wait to receive the response of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user (default value is 120 seconds).
- **Request Zlib Headers.** Sends request data to the server with the **zlib** compression library headers. By default, requests sent to the server include the **zlib** headers. This option lets you emulate non-browser applications that do not include **zlib** headers in their requests. To exclude these headers, set this option to **No** (default is Yes).
- **Accept Server-Side Compression.** Indicate to the server that the replay can accept compressed data. The available options are: **None** (no compression), **gzip** (accept gzip compression), **gzip, deflate** (accept gzip or deflate compression), and **deflate** (accept deflate compression). Note that by accepting compressed data, you may significantly increase the CPU consumption. The default is to accept **gzip, deflate** compression.

General

- **DNS caching.** Instructs the Vuser to save a host's IP addresses to a cache after resolving its value from the Domain Name Server. This saves time in subsequent calls to the same server. In situations where the IP address changes, as with certain load balancing techniques, be sure to disable this option to prevent Vuser from using the value in the cache (enabled by default).

- **Convert from/to UTF-8.** Converts received HTML pages and submitted data from and to UTF-8. You enable UTF-8 support in the recording options. For more information, see "Web, Wireless, and Oracle NCA Recording Options" on page 1168 (No by default).
- **Step timeout caused by resources is a warning.** Issues a warning instead of an error when a timeout occurs due to a resource that did not load within the timeout interval. For non-resources, VuGen issues an error (disabled by default).
- **Parse HTML Content-Type.** When expecting HTML, parse the response only when it is the specified content-type: **HTML**, **text\html**, **TEXT** any text, or **ANY**, any content-type. Note that text/xml is not parsed as HTML. The default is **TEXT**.

The timeout settings are primarily for advanced users who have determined that acceptable timeout values should be different for their environment. The default settings should be sufficient in most cases. If the server does not respond in a reasonable amount of time, check for other connection-related issues, rather than setting a very long timeout which could cause the scripts to wait unnecessarily.

- **Step download timeout (sec).** The time that the Vuser will wait before aborting a step in the script. This option can be used to emulate a user behavior of not waiting for more than x seconds for a page.
- **Network buffer size.** Sets the maximum size of the buffer used to receive the HTTP response. If the size of the data is larger than the specified size, the server will send the data in chunks, increasing the overhead of the system. When running multiple Vusers from the Controller, every Vuser uses its own network buffer. This setting is primarily for advanced users who have determined that the network buffer size may affect their script's performance. The default is 12K bytes. The maximum size is 0x7FFF FFFF.
- **Print NTLM information.** Print information about the NTLM handshake to the standard log.
- **Print SSL information.** Print information about the SSL handshake to the standard log.

- **Max number of error matches issued as ERRORS.** Limits the number of error matches issued as ERRORS for content checks using a LB or RB (left boundary or right boundary). This applies to matches where a failure occurs when the string is found (Fail=Found). All subsequent matches are listed as informational messages. The default is 10 matches.
- **Maximum number of META Refresh to the same page.** The maximum number of times that a META refresh can be performed per page. The default is 2.
- **ContentCheck values in UTF-8.** Store the values in the ContentCheck XML file in UTF-8.

Authentication

- **Fixed think time upon authentication retry (msec).** Automatically adds a think time to the Vuser script for emulating a user entering authentication information (username and password). This think time will be included in the transaction time (default is 0).
- **Disable NTLM2 session security.** Use full NTLM 2 handshake security instead of the more basic NTLM 2 session security response (default is No).
- **Use Windows native NTLM implementation.** Use the Microsoft Security API for NTLM authentication instead of the indigenous one (default is No).
- **Enable integrated Authentication.** Enable Kerberos-based authentication. When the server proposes authentication schemes, use **Negotiate** in preference to other schemes (default is No).
- **Induce heavy KDC load.** Do not reuse credentials obtained in previous iterations. Enabling this setting will increase the load on the KDC (Key Distribution Server). To lower the load on the server, set this option to **Yes** in order to reuse the credentials obtained in previous iterations. This option is only relevant when Kerberos authentication is used (default is No).

Log

- **Print buffer line length.** Line length for printing request/response header/body and/or JavaScript source, disabling wrapping.

- **Print buffer escape only binary zeros.**
  - **Yes.** Escape only binary zeros when printing request/response headers/body and/or JavaScript source.
  - **No**. Escape any unprintable/control characters.

Web (Click and Script) Specific

- **General**
  - **Home Page URL.** The URL of the home page that opens with your browser (default is about:blank).
  - **DOM-based snapshots.** Instructs VuGen to generate snapshots from the DOM instead of from the server responses (**Yes** by default).
  - **Charset conversions by HTTP.** Perform charset conversions by the 'Content-Type:....; charset=...' HTTP response header. Overrides 'Convert from /to UTF-8.'
  - **Reparse when META changes charset.** Reparse HTML when a META tag changes the charset. Effective only when **Charset conversions by HTTP** is enabled. **Auto** means reparsing is enabled only if it used in the first iteration.
  - **Fail on JavaScript error.** Fails the Vuser when a JavaScript evaluation error occurs. The default is No, issuing a warning message only after a JavaScript error, but continuing to run the script.
  - **Initialize standard classes for each new window project.** When enabled, the script—the src compiled script, will not be cached.
  - **Ignore acted on element being disabled.** Ignore the element acted on by a Vuser script function being disabled.
- **Timers**
  - **Optimize timers at end of step.** When possible, executes a setTimeout/setInterval/<META refresh> that expires at the end of the step before the expiration time (default is Yes).

- **Single setTimeout/setInterval threshold (seconds).** Specifies an upper timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, these methods will not invoke the functions that are passed to them. This emulates a user waiting a specified time before clicking on the next element (default is 5 seconds).
- **Accumulative setTimeout/setInterval threshold (seconds).** Specifies a timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, additional calls to window.setTimeout and window.setInterval will be ignored. The timeout is accumulative per step (default is 30 seconds).
- **Reestablish setInterval at end of step. 0** = No; **1** = Once; **2** = Yes.

- **History**
  - **History support.** Enables support for the window.history object for the test run. The options are Enabled, Disabled, and Auto. The Auto option (default) instructs Vusers to support the window.history object only if it was used in the first iteration. Note that by disabling this option, you improve performance.
  - **Maximum history size.** The maximum number of steps to keep in the history list (default is 100 steps).
- **Navigator Properties**
  - **navigator.browserLanguage.** The browser language set in the navigator DOM object's **browserLanguage** property. The default is the recorded value. Scripts created with older recording engines, use **en-us** by default.
  - **navigator.systemLanguage.** The system language set in the navigator DOM object's **systemLanguage** property. The default is the recorded value. Scripts created with older recording engines, use **en-us** by default.
  - **navigator.userLanguage.** The user language set in the navigator DOM object's **userLanguage** property. The default is the recorded value. Scripts created with older recording engines, use **en-us** by default.
- **Screen Properties**

- **screen.width** Sets the width property of the screen DOM object in pixels (default is 1024 pixels).
- **screen.height** Sets the height property of the screen DOM object in pixels (default is 768 pixels).
- **screen.availWidth** Sets the availWidth property of the screen DOM object in pixels (default is 1024 pixels).
- **screen.availHeight.** Sets the availHeight property of the screen DOM object in pixels (default is 768 pixels).

- **Memory Management**
  - **Default block size for DOM memory allocations.** Sets the default block size for DOM memory allocations. If the value is too small, it may result in extra calls to malloc, slowing the execution times. Too large a block size, may result in an unnecessarily big footprint (default is 16384 bytes).
  - **Memory Manager for dynamically-created DOM objects. Yes**—Use the Memory Manager for dynamically-created DOM objects. **No—**Do not use the Memory Manager, for example when multiple DOM objects are dynamically created in the same document as under SAP. **Auto**—Use the protocol recommended (default Yes for all protocols except for SAP).
  - **JavaScript Runtime memory size (KB).** Specifies the size of the JavaScript runtime memory in kilobytes (default is 256 KB).
  - **JavaScript Stack memory size (KB).** Specifies the size of the JavaScript stack memory in kilobytes (default is 32 KB).

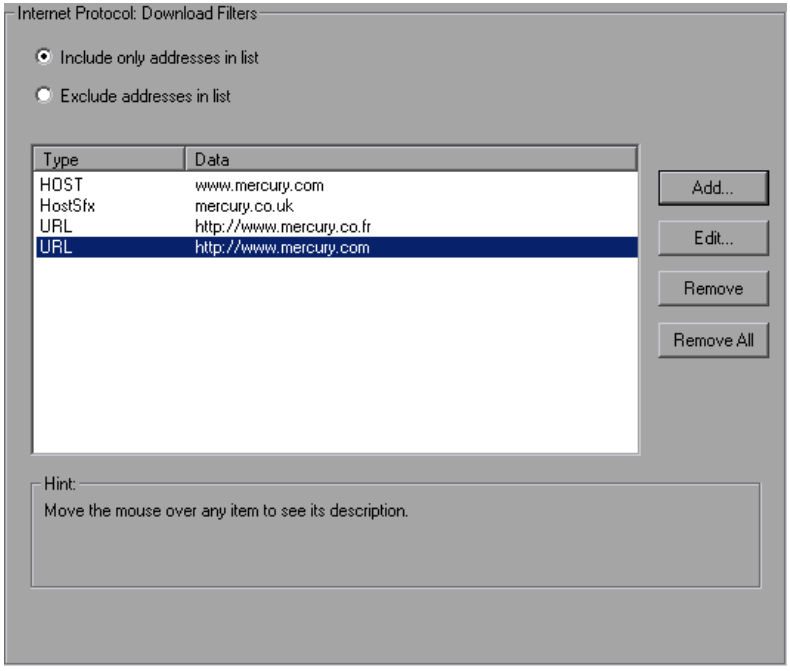
Filtering Web Sites

You can specify the Web sites from which Vusers should download resources during replay. You can indicate either the sites to exclude or the sites to include. You control the allowed or disallowed sources, by specifying a URL, host name, or host suffix name.

A **URL** is the complete URL address of a Web site, beginning with http:// or https://. **Host** is the name of the host machine with its domain, such as www.hp.com.

Host suffix is the common suffix for several host names, such as hp.com. This is useful where you have several Web sites on a common domain.

If you specify the sites to exclude, VuGen downloads resources from all Web sites except for those specified in the list. If you specify the sites to include, VuGen filters out resources from all Web sites except for those in the Include list.

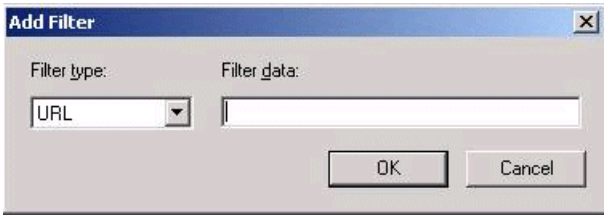


To create a list of filtered Web sites:

1 Click the **Internet Protocol:Download Filters** node.

2 Select the desired option: **Include only addresses in list** or **Exclude addresses in list**.

3 Add entries to the list. To add an entry, click **Add**. The Add filter dialog box opens.



Select a filter type: **URL**, **Host**, or **Host Suffix**, and enter the filter data, such as a URL. When entering a URL, make sure to enter a complete URL beginning with **http://** or **https://**. Click **OK**.

4 To edit an entry, select it and click **Edit**.

5 To delete and entry, select it and click **Remove**. To delete all entries, click **Remove All**.

Obtaining Debug Information

When you run a Vuser script, the execution information is displayed in the Output window or log file. You control the amount of information sent to the Output window and log files, using the **Log** node of the General run-time settings. For more information, see "Configuring the Log Run-Time Settings" on page 1260.

Debug information includes:

- log information
- transaction failures
- the connection status with the gateway—connecting, disconnecting, and redirecting. (WAP only)

To obtain more information for debugging, edit the **default.cfg** file. Locate the **WEB** section and set the **LogFileWrite** flag to **1**. The resulting trace file documents all events in the execution of the script.

When performing load testing, make sure to clear the **LogFileWrite** flag to prevent the Vusers from wasting resources by creating a large trace file.

Performing HTML Compression

Browsers that support HTTP 1.1 can decompress HTML files. The server compresses the files for transport, substantially reducing the bandwidth required for the data transfer. You can enable compression automatically or manually.

To automatically enable compression in VuGen, use the **Internet Protocol** > **Preferences** node of the Run-Time settings. Click **Options** to open the Advanced Options and enable the **Accept Server-Side compression** option. Note that this option is enabled by default. For more information, see "Additional Options for Internet Preferences" on page 1290.

To manually add compression, enter the following function at the beginning of the script:

```
web_add_auto_header("Accept-Encoding", "gzip");
```

To verify that the server sent compressed data, search for the string **Content -Encoding: gzip** in the section of the server's responses of the Execution log. The log also shows the data size before and after decompression.

Compression has a greater effect on large data transfers—the larger the data, the greater effect the compression will have. When working with larger data, you can also increase the network buffer size (see the Network Buffer Size option) to get the data in single chunks.

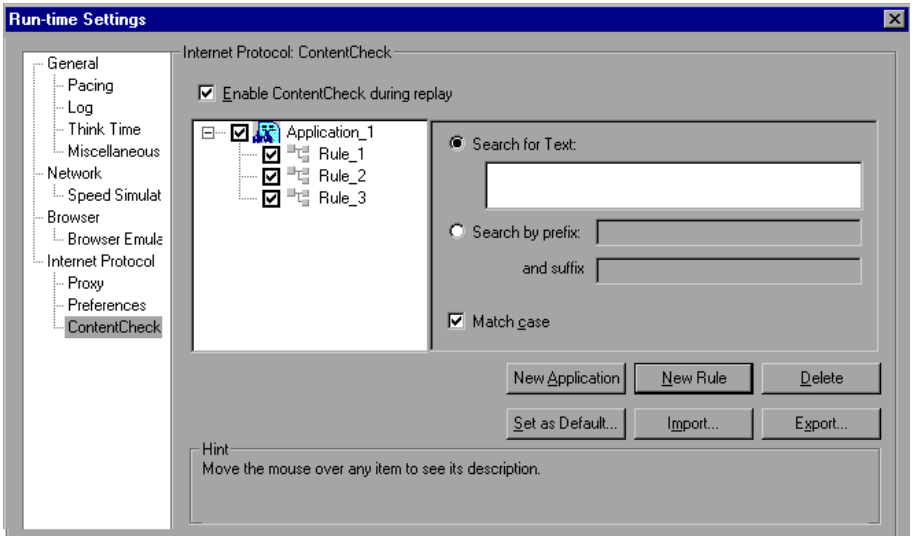
Checking Web Page Content

VuGen's Content Check mechanism allows you to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text **ASP Error**. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay. Note that VuGen searches the body of the pages—not the headers.

You use the **Internet Protocol:ContentCheck** Run-Time setting to specify the content for which you want to search. You can define content for several applications with multiple rules. The following sections discuss:

- Understanding Content Rules
- Defining ContentCheck Rules



Understanding Content Rules

You use the ContentCheck run-time options to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text **ASP Error**. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay. Note that VuGen searches the body of the pages—not the headers.

➤ **Enable ContentCheck during replay.** Enable content checking during replay (enabled by default). Note that even after you define applications, you can disable it for a specific test run, by disabling this option.

Rule Information

This right pane contains the matching criteria for the text you want to find. You can specify either the actual text or a prefix and suffix of the text.

➤ **Search for Text.** The text of the string for which you want to search.

➤ **Search by Prefix and Suffix.** The prefix and suffix of the string for which you want to search.

➤ **Match case.** Perform a case sensitive search.

➤ **Search JavaScript alert box text.** Only search for text within JavaScript alert boxes (Web (Click and Script), PeopleSoft Enterprise, and Oracle Web Applications 11i Vusers only).

Adding and Removing Applications and Rules

➤ **New Application.** Automatically adds a new application to the list of applications in the left pane. The default name is Application_**index**, beginning with **Application_1**. After you create a new application, click **New Rule** to add a rule to this application. To modify the name of an application, double-click on it.

- **New Rule.** Displays the rule criteria in the right pane, allowing you to enter a new rule for the currently selected application. The rules are stored with the script in standard xml files. You can export your rule files and share them with other users or import them to other machines.
- **Delete.** Deletes the selected rule or application.

Importing and Exporting Rules

- **Import/Export.** Imports or exports a rule file. The rule file with an **xml** extension, stores the applications and rules. You can export the file to use on other machines. You can also import other rule files. If you import a rule and the selected rule conflicts with an existing rule, VuGen issues a warning indicating that it is a Conflicting Rule. You can then merge the rules you created on a former script with the one you are importing or overwrite the current rules. When you click Export, VuGen opens the Choose Application to Export dialog box.

Setting Rules as Default

- **Set as Default.** There are three types of rules for Content Checks: **Installation**, **Default**, and **per script**. Installation rules are provided automatically during installation of the product. Default rules, apply to all scripts executed on your machine. The per script rules are the ones defined for the current script. When you modify or add rules, these changes only apply to the current script. To instruct VuGen to add a rule to the list of Default rules so that it will apply to all scripts on that machine, click **Set as Default**.

When working on multiple scripts, or when performing a product upgrade, a conflict may arise between the default rules and the script rules. VuGen asks you if you want to merge the rules. When you merge the rules (recommended), the rule is added to the list of rules for the application.

This action only effects applications that are enabled in the Application list (the left pane). If no applications were marked as Enabled in the current script, no application will be marked as Enabled in the Defaults file. Click **Yes** to overwrite the Defaults file. Click **No** to cancel the operation and retain the original Defaults file.

The rules are stored in standard xml files. You can export your rule files and share them with other users or import them to other machines.

When you click Set as Defaults (and confirm the overwriting), VuGen performs the following actions:

1. Marks all applications in the Defaults File as **Disabled**.
2. For applications marked as **Enabled** in the current script, it performs a merge or copy, depending on whether the application exists. If the application exists, it merges the rules of the current script with those of the Defaults file. If the application did not exist in the Defaults file, then VuGen just copies the rules to the Defaults file.
3. Marks the applications that were enabled in the script, as **Enabled** in the Defaults file. If no application is marked as **Enabled** in the current script, no application will be marked as **Enabled** in the Defaults file.

Use Defaults

Imports rules from the Defaults file. When you click this button, VuGen opens a dialog box with a list of the applications and their default settings. You can import these rules or modify them. If this conflicts with one of the existing rules, VuGen issues a warning indicating that it is a Conflicting Rule. You can also merge the rules defined in the Defaults file with the ones currently defined.

To use the default settings for all of your applications, click **Use Defaults** which imports the definitions from the Defaults file. It opens a dialog box with a list of the applications and their default settings. You can import these definitions or modify them. If this conflicts with one of the rules, VuGen issues a warning indicating that it is a Conflicting Rule. You can merge or overwrite the rules defined in the Defaults file with the active ones.

Defining ContentCheck Rules

You use the **Internet Protocol:ContentCheck** node in the Run-Time Setting tree, to define the rules for checking Web page content.

To define a ContentCheck rule:

1. Open the Run-Time settings and select the **Internet Protocol:ContentCheck** node.
2. Select the **Enable ContentCheck during replay** option.

3. Click **New Application** to add a new entry to the list of applications whose content to check.
4. Click **New Rule** to add rules for existing applications. Each application server may have one or more rules. Enable or disable the relevant rules by clearing or selecting the check boxes adjacent to the rule in the left pane.
5. To search for the actual text string, select **Search for Text** and specify the text for which you want to search. To obtain the best results, be as specific as possible. For example, do not use the term **Error**, rather **ASP Error** or text specific to the application.
6. To search for the text preceding and following your string, select **Search by Prefix** and specify the prefix and suffix.
7. To indicate a case sensitive search, select the **Match case** check box.
8. To set a rule as a default, indicating that it should apply to all scripts on that machine, select the rule or application and click **Set as Default**.
9. To export the rule file click **Export** and specify a save location.
10. To import a rule file, click **Import** and locate the file.
11. To remove an application or rule, select it and click **Delete**.
12. To use the default settings for all of your applications, click **Use Defaults.** A dialog box opens with a list of the applications and their default settings. You can overwrite or merge the rules if there are conflicts.

81

Run-Time Settings for Selected Protocols

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.

This chapter includes:

RDP Run-Time Settings

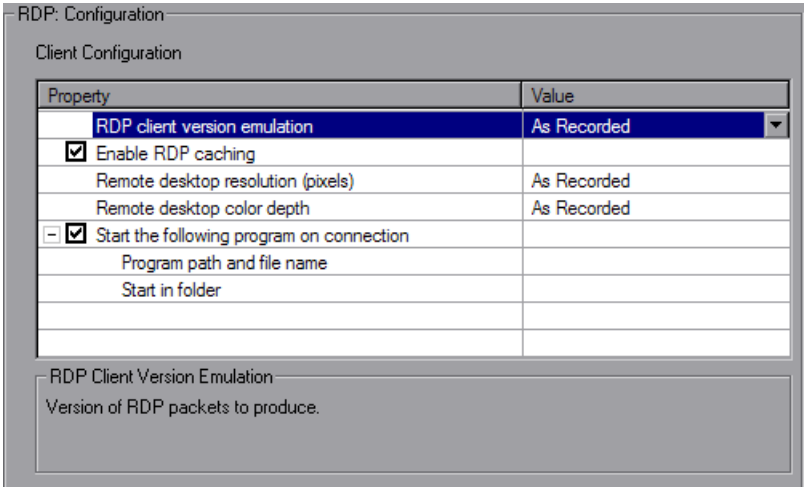
After creating an RDP Vuser script, you set the run-time settings. These settings let you control the behavior of the Vuser when running the script.

You can set the RDP-specific run-time settings in the following areas:

- RDP Configuration Run-Time Settings
- RDP Synchronization Run-Time Settings
- RDP Advanced Run-Time Settings
- RDP Agent Run-Time Settings

RDP Configuration Run-Time Settings

You use the RDP Configuration settings to set the behavior of the (virtual) RDP client.

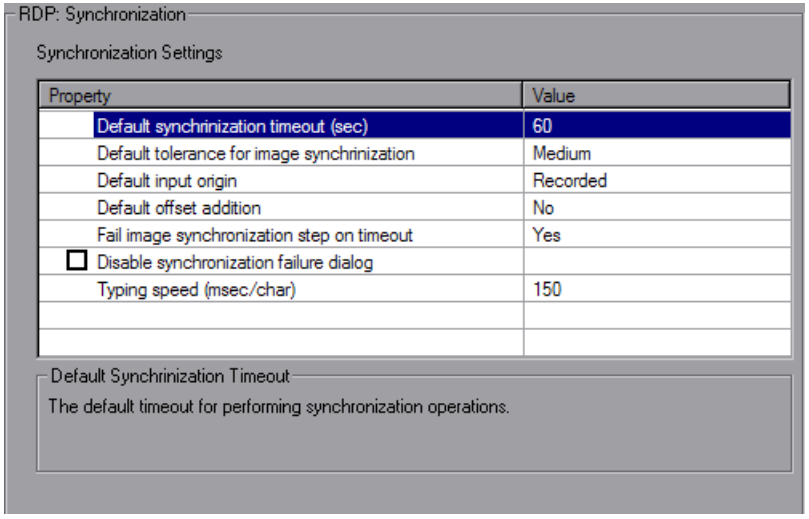


- **RDP Client Version Emulation.** The version of RDP packets to produce during replay: **As Recorded**, or a specific version number.
- **Enable RDP caching.** Support data caching orders in RDP (enabled by default).
- **Remote desktop resolution (pixels).** The size of the window in which the applications are run: **As Recorded**, or a specific size.

- **Remote desktop color depth.** The color depth settings for the replay: **As Recorded**, or a specific depth.
- **Start the following program on connection.** Open RDP connection to invoke the specified application. Specify the following information: **Program path and file name** and optionally, **Start in folder**.

RDP Synchronization Run-Time Settings

RDP Synchronization settings indicate the default timing values for various functions.



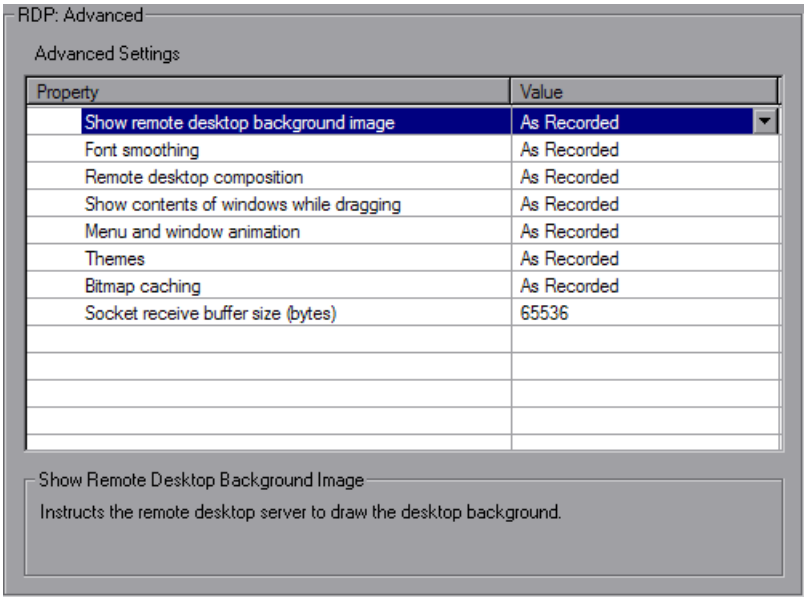
- **Default synchronization timeout (sec)**. The time in seconds to wait for synchronization operations. Enter a value between 0 and 1000. The default value is 60.
- **Default tolerance for image synchronization.** The tolerance level for performing synchronization on images. Select one of the options: **Exact**, **Low**, **Medium** (default), or **High**. **High** has the most tolerance for changes and mismatches. **Low** requires a match of approximately 95 percent, **Medium** requires a match of approximately 85 percent, **High** requires a match of approximately 70 percent, and **Exact** requires an 100 percent match.
- **Default input origin.** The default origin for input operations:

- **Recorded.** Uses coordinates for all input operations with a non-specified input origin. This is the default selection.
- **Synched.** Adds the most recent offsets saved at one of the previous synchronization functions to the recorded coordinates of each input operation with a non-specified input origin.

- **Default offset addition.** Saves the offset of images that moved during synchronization for all subsequent functions (**No** by default).
- **Fail image synchronization step on timeout.** Instructs Vusers how to proceed when images are not found during synchronization. **Yes** (default) sets a Fail status and Vusers follow the Continue on Error setting. **No** returns an LR_NOT_FOUND flag, the step reports a warning and the script continues.
- **Disable synchronization failure dialog.** When selected, it prevents the Synchronization Failure Dialog box from opening (not selected by default).
- **Typing speed (msec/char)**. The time in milliseconds for sending consecutive characters in keyboard commands. Enter a value between 0 and 1000. The default value is 150.

RDP Advanced Run-Time Settings

You can edit advanced RDP Run-Time settings in the Advanced Settings dialog box.

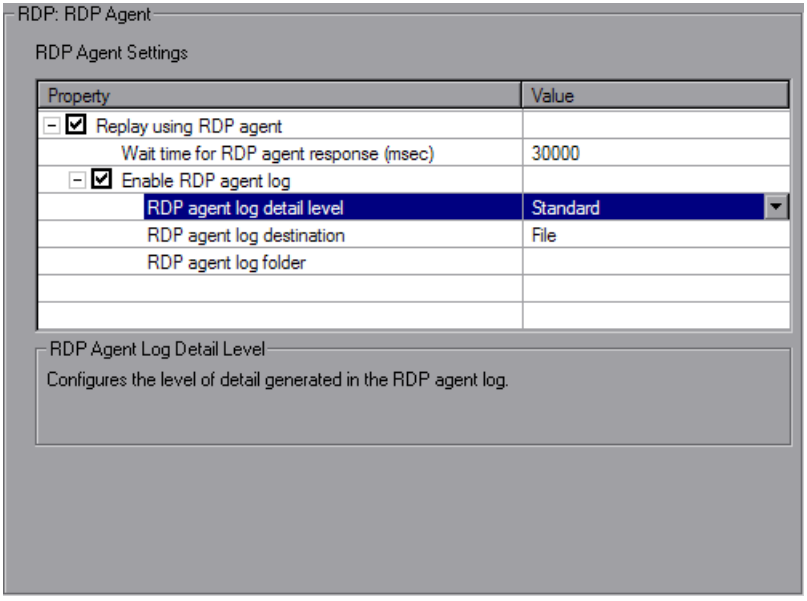


- **Show remote desktop background image.** Allows you to run the remote desktop application without displaying the desktop background image on the remote desktop. Disabling this setting can save system resources on the remote desktop server.
- **Font smoothing.** Allows the remote desktop server to use font smoothing. Disabling this setting can save system resources on the remote desktop server.
- **Remote desktop composition.** Enables remote desktop composition.
- **Show contents of window while dragging.** Shows the contents of windows while they are being dragged. Disabling this setting can save system resources on the remote desktop server.
- **Menu and window animation.** Allows the remote desktop server to animate menus and windows. Disabling this setting can save system resources on the remote desktop server.

- **Themes.** Allows the remote desktop server to use Windows themes. Disabling this setting can save system resources on the remote desktop server.
- **Bitmap caching.** Allows the remote desktop server to use bitmap caching. Enabling this setting can save system resources on the remote desktop server.
- **Socket receive buffer size (bytes).** The number of bytes to allocate for the socket's receive buffer. If the buffer is too small, it can fill up causing the server to disconnect. If the buffer is too large, it uses more local system resources (memory).

RDP Agent Run-Time Settings

The Agent run-time settings control the way the RDP agent for Microsoft Terminal Server functions with VuGen during replay.



You can set the following options:

- **Use RDP agent.** Instructs VuGen to use RDP agent during recording, then generates script using information gathered by the RDP agent during the recorded session. LoadRunner RDP agent must be installed on the server.
- **Enable RDP agent log.** Enables the RDP agent log. This feature should be used only for debugging purposes.
  - **RDP agent log detail level.** Configures the level of detail generated in the RDP agent log with **Standard** being the lowest level of detail and **Extended Debug** being the highest level of detail.
  - **RDP agent log destination.** Configures the destination of the RDP agent log data. **File** saves the log messages only on the remote server side. **Stream** sends the log messages to the Vugen machine. **FileAndStream** sends the log messages to both destinations.
  - **RDP agent log folder.** The folder path on the remote server that the RDP agent log file will be generated in. If none is specified and the agent log destination was set to **File**, the log is saved in the temp folder of the user on the server.

Citrix Run-Time Settings

After creating a Citrix Vuser script, you set the run-time settings. These settings let you control the behavior of the Vuser when running the script. Your Citrix run-time settings in the **Configuration** node should correspond to the properties of your Citrix client. These settings will influence the load on the server. To view the connection properties, select the icon representing the ICA connection in the Citrix Program Neighborhood, and select **Properties** from the right-click menu. Select the **Default Options** tab.

Note: Citrix Vusers do not support IP spoofing.

To set the General Run-time settings, see Chapter 79, "Configuring Run-Time Settings." To set the Speed Emulation properties, see Chapter 80, "Configuring Network Run-Time Settings."

You can set the Citrix-specific run-time settings in the following areas:

- Citrix Configuration Run-Time Settings
- Citrix Synchronization Run-Time Settings

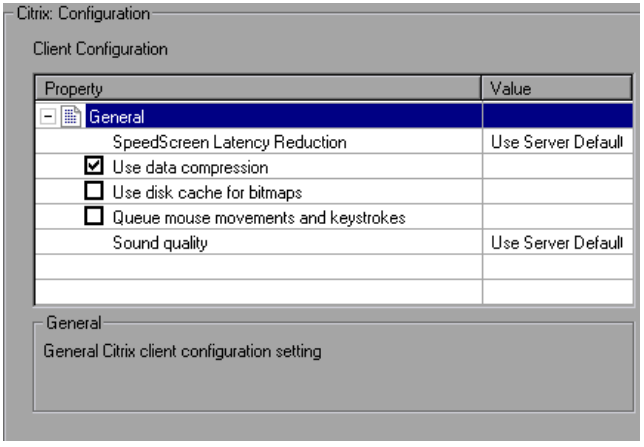
Citrix Configuration Run-Time Settings

The configuration settings relate to the screen latency, data compression, disk cache, and queuing of mouse movements.

To set the Configuration Run-Time Settings:

1 Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the VuGen toolbar, or select **Vuser** > **Run-Time Settings**.

2 Select the **Citrix:Configuration** node. Specify the **General** properties:



Set the desired client configuration options:

- **SpeedScreen Latency Reduction**. The mechanism used to enhance user interaction when the network speed is slow. You can turn this mechanism **on** or **off**, depending on the network speed. The **auto** option turns it on or off based on the current network speed. If you do not know the network speed, set this option to **Use Server Default** to use the machine's default.

- **Use data compression.** Instructs Vusers to compress the transferred data. To enable this option, select the check box to the left of the option; to disable it, clear the check box. You should enable data compression if you have a limited bandwidth (enabled by default).
- **Use disk cache for bitmaps.** Instructs Vusers to use a local cache to store bitmaps and commonly-used graphical objects. To enable this option, select the check box to the left of the option; to disable it, clear the check box. You should enable this option if you have a limited bandwidth (disabled by default).
- **Queue mouse movements and keystrokes.** Instructs Vusers to create a queue of mouse movements and keystrokes, and send them as packets to the server less frequently. This setting reduces network traffic with slow connections. Enabling this option makes the session less responsive to keyboard and mouse movements. To enable this option, select the check box to the left of the option; to disable it, clear the check box (disabled by default).
- **Sound quality.** Specifies the quality of the sound: **Use server default**, **Sound off**, **High sound quality**, **Medium sound quality**, or **Low sound quality**. If the client machine does not have a 16-bit Sound Blaster-compatible sound card, select **Sound Off**. With sound support enabled, you will be able to play sound files from published applications on your client machine.

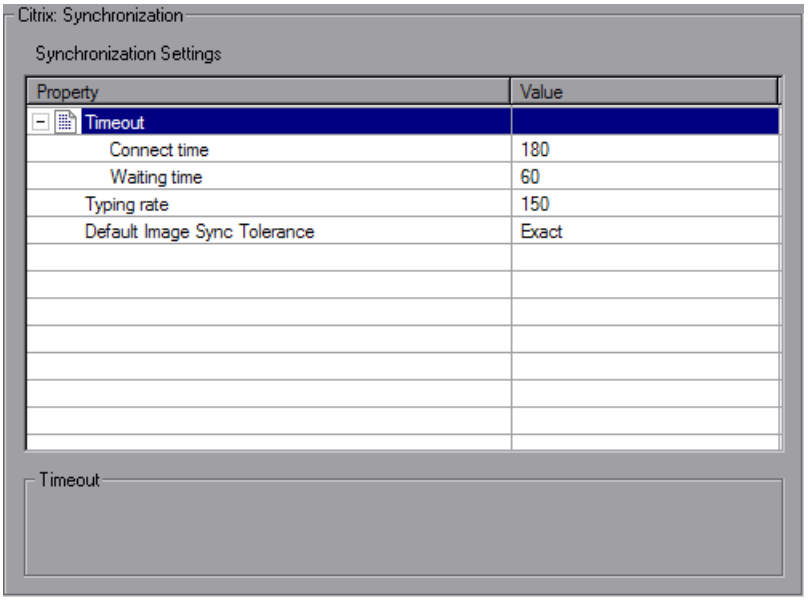
Citrix Synchronization Run-Time Settings

The synchronization settings relate to the connection and waiting times.

To set the synchronization run-time settings:

1 Click the **Run-Time Settings** button on the VuGen toolbar, or select **Vuser > Run-Time Settings**.

2 Select the **Citrix:Synchronization** node.



3 Indicate the **Connect Time**, the time (in seconds) to wait idly at an established connection before exiting. The default is 180 seconds.

4 Indicate the **Waiting Time**, the time (in seconds) to wait idly at a synchronization point before exiting. The default is 60 seconds.

To set the waiting time for a specific section of the script, click **Insert** > **New Step** and insert a **Set Waiting Time** step. The new waiting time applies from the point of insertion until the end of the script or the next **Set Waiting Time** step.

5 Specify a **Typing rate**, the delay (in milliseconds) between keystrokes.

6 Indicate the **Default Image Sync Tolerance** level. This setting controls the level of equality two images must share to be considered synchronized.

- **Exact.** (default setting) Must have a 100% match.
- **Low.** Must have a 95 % match.
- **Medium.** Must have an 85% match.

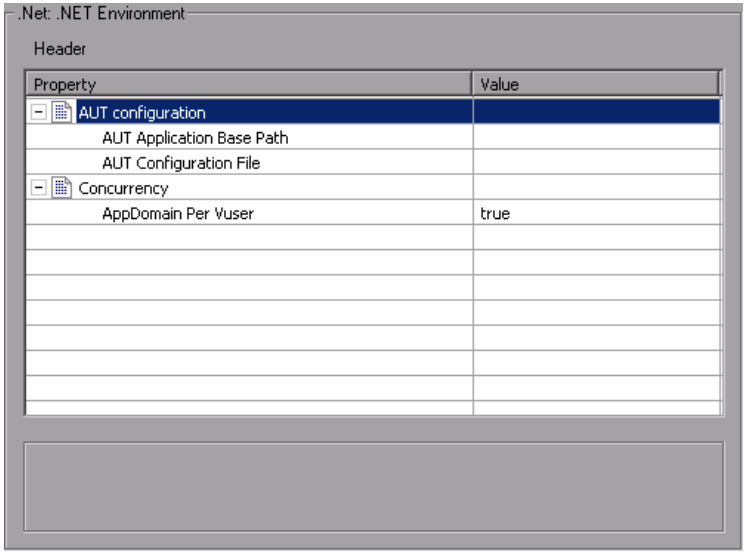
➤ **High**. Must have a 70% match.

For more information on synchronization, see "Synchronizing Replay" on page 528.

7 Click **OK** to save the settings and close the dialog box.

.NET Environment Run-Time Settings

Before running your Microsoft .NET Vuser script, you can specify the .NET environment settings from the Run-Time Settings dialog box.



You can also set general run-time settings for your Microsoft .NET script for configuring the pacing and iteration options. For more information, see Chapter 79, "Configuring Run-Time Settings."

AUT Configuration

AUT Application Base Path. The AUT (Application Under Test) base directory from which DLLs are loaded during replay. By default, during recording, all of the necessary DLLs are stored in the script's directory. Use this option to specify the location of any missing DLL files for the AUT. This is usually the installation path of the recorded application. Note that the AUT must be installed on the machine running the script. If you leave this box empty, VuGen uses the local script\bin directory as the application base directory during replay.

AUT Configuration File. The file name of the recorded application's configuration file. VuGen copies the AUT configuration file to the script\bin directory and loads the locally saved file. To specify a different location, use a full path. If you only specify a file name, and the file is not in the script\bin folder, VuGen loads it from the App base directory.

Concurrency

➤ **AppDomain Per Vuser.** Enables execution of each Vuser in a separate app domain (true by default). Running Vusers in separate App Domains enables each Vuser to execute separately without sharing static variables and prevents locking between them.

ADO.NET providers deploy a feature called **connection pooling** which can significantly influence load test accuracy. Whenever only one app domain is used for all Vusers, connection pooling is turned on—.NET Framework keeps the database connections open and tries to reuse them when a new connection is requested. Since many Vusers are executed in the context of a single application domain, they may interfere with one another. Their behavior will not be linear and that may decrease their accuracy. The default setting, **true**, allocates a separate connection pool for each Vuser. This means that there is connection pooling in the scope of each Vuser, but the Vusers will not interfere with one another. This setting provides more accuracy, but lower scalability.

If you disable this option, you need to manually disable connection pooling for the database.

The following table describes how to manually disable connection pooling:

Provider	Option
.NET Framework Data Provider for SQL Server	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for Oracle	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for ODBC	Connection pooling is managed by an ODBC Driver Manager. To enable or disable connection pooling, use the ODBC Data Source Administrator (found in Control Panel or the Administrative Tools folder). The **Connection Pooling** tab allows you to specify connection pooling parameters for each of the installed ODBC drivers.
.NET Framework Data Provider for OLE DB	"OLE DB Services=-2"
Oracle Data Provider for .NET	"pooling=false"
Adaptive Server Enterprise ADO.NET Data Provider	"Pooling=False"

To specify .NET resources:

1. Open the run-time setting—press F4 or select **Vuser** > **Run-Time Settings**.
2. Click on the **.NET Environment** node in the left pane.
3. Set the base folder of the DLLs in the **AUT Application Base Path** box.
4. Set the path of the recorded application in the **AUT Configuration File** box.
5. The recommended setting for **AppDomain Per Vuser** is true, the default.

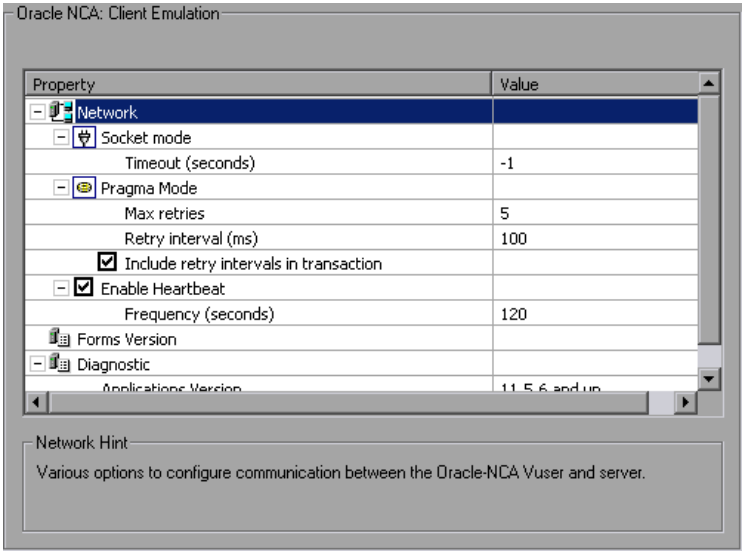
Oracle NCA Run-Time Settings

Before running your script, you can set the run-time settings to allow the script to accurately emulate a real user. For information on the general run-time settings for all protocols, such as think time, pacing, and logging, see Chapter 79, "Configuring Run-Time Settings." For network speed related settings, see Chapter 80, "Configuring Network Run-Time Settings."

The following section describes the run-time settings specific to Oracle NCA Vusers. These run-time setting allow you to indicate the communication parameters.

Configuring Oracle NCA Client Emulation Run-Time Settings

You can configure several network settings to accurately emulate an Oracle NCA client.



You can set the following options:

Socket Mode

The communication to and from the client is performed on a socket level—not on the higher HTTP level.

Timeout (seconds): The time that an Oracle NCA Vuser waits for a response from the server. The default value of -1 disables the timeout and the client waits indefinitely.

Pragma Mode

In Pragma mode, communication is carried out in the Oracle-defined Pragma mode. This communication level, above the HTTP and Servlet levels, is characterized by the periodic sending of messages. In this mode, the client recognizes that the server cannot respond with data immediately. The server sends messages at given intervals until it is able to send the requested data.

- **Max Retries**. Indicates the maximum number of **IfError** messages the client will accept from the server before issuing an error. **IfError** messages are the periodic messages the server sends to the client, indicating that it will respond with the data as soon as it is able.
- **Retry Interval.** Defines the interval between retries in the case of **IfError** messages.
- **Include retry intervals in transaction.** Includes the interval between retry time, as part of the transaction duration time.

For information about recording in Pragma mode, see "Recording in Pragma Mode" on page 909.

Heartbeat

You can enable or disable the heartbeat sent to the Oracle server. The heartbeat verifies that there is proper communication with the server. If you are experiencing a heavy load on the Oracle NCA server, disable the heartbeat. If you enable the heartbeat, you can set the frequency of how often heartbeat messages are sent to the server.

➤ **Enable Heartbeat.** By default, a heartbeat signal is sent to the server. To disable it, clear the check box.

➤ **Frequency.** The frequency of the heartbeat signal. The default is 120 seconds.

Forms

You can specify the version of the Oracle Forms server detected during recording.

➤ **Version.** Modify this setting only if the server was upgraded since the recording.

Diagnostic

This section lets you provide information about diagnostic modules for the database layer of Oracle Applications.

➤ **Application version.** The version of Oracle Application. This option is relevant when using Oracle Application—not a custom Oracle NCA application. It is only required when using Oracle database breakdown.

To set the Client Emulation settings:

1 Open the Run-Time Settings dialog box. Select **Vuser** > **Run-Time Settings** or click the **Run-Time Settings** button on the VuGen toolbar.

2 Select the **Oracle NCA:Client Emulation** node from the Run-Time settings tree.

3 Set the network timeout value in seconds. To instruct the client to wait indefinitely for a server response, use the default value of -1.

4 When working in Pragma mode, specify the number of retries **Max Retries**, (**IfError** messages) for the client to accept before issuing an error. The default is 5.

5 To enable the sending a a heartbeat to the Oracle NCA server, select the **Enable Heartbeat** option. In the next line, specify a frequency in seconds for the sending of the heartbeat. The default is 120 seconds.

6 Click **OK** to accept the settings and run the script.

SAPGUI Run-Time Settings

After creating and enhancing your SAPGUI Vuser script, you configure its run-time settings and run it from VuGen to check its functionality. Run-Time settings let you control the Vuser behavior during replay. You configure these settings before running the Vuser script. You can set both general and SAPGUI-specific run-time settings.

The general settings include the run logic, pacing, logging, think time, and performance preferences. For information about the general run-time settings, see Chapter 79, "Configuring Run-Time Settings." For SAPGUI-specific settings, see the following sections.

Once you configure the Run-Time settings, you save the Vuser script and run it from VuGen to verify that it runs correctly. For details about running the Vuser script as a standalone test, see Chapter 9, "Running Vuser Scripts in Standalone Mode."

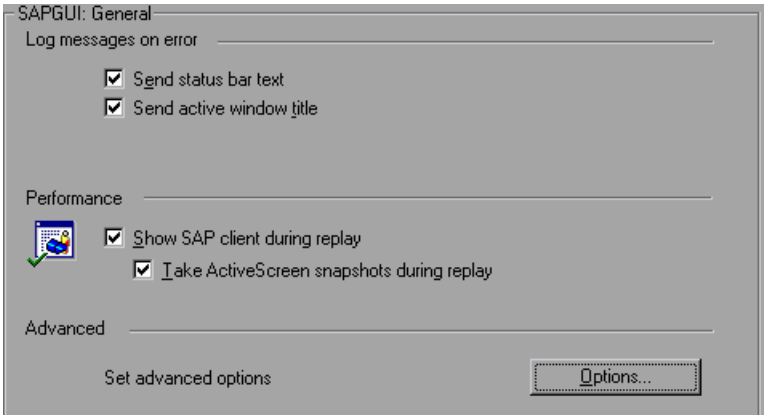
After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

You can configure the SAPGUI specific Run-Time settings in the following areas:

- SAPGUI General Run-Time Settings
- SAPGUI Advanced Run-Time Settings

SAPGUI General Run-Time Settings

General run-time settings let you set the general settings for a SAPGUI Vuser script. VuGen uses these settings when running the script.



The Log run-time settings specify the information a Vuser sends to the Execution log whenever an error occurs.

- **Send status bar text.** Send the text from the status bar to the log file.
- **Send active window title.** Send the active window title text to the log file.

The Performance run-time settings allow you to indicate whether or not to display the SAP client during replay.

- **Show SAP Client during replay.** Shows an animation of the actions in the SAP client during replay. The benefit of displaying the user interface (UI) is that you can see how the forms are filled out and closely follow the actions of the Vuser. This option, however, requires additional resources and may affect the performance of your load test.

➤ **Take ActiveScreen snapshots during replay.** Captures replay snapshots with the Control ID information for all active objects. ActiveScreen snapshots differ from regular ones, in that they allow you to see which objects were recognized by VuGen in the SAPGUI client. As you move your mouse across the snapshot, VuGen highlights the detected objects. You can then add new steps to the script directly from within the snapshot. It also allows you to add steps interactively from within the snapshot for a specific object. For more information, see "Inserting Steps Interactively into a SAPGUI Script" on page 930.

Advanced options let you set a timeout for the **SAPfewgsvr.exe** process, save a snapshot on error, and configure VuGen to use SAPlogon during replay. For more information, see "SAPGUI Advanced Run-Time Settings" on page 1323.

To set the SAPGUI Run-Time Settings:

1 Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the VuGen toolbar, or select **Vuser** > **Run-Time Settings**.

2 Select the **SAPGUI:General** node.

3 In the **Log messages on error** section, select one or more message sources: **Send status bar text** or **Send active window title**.

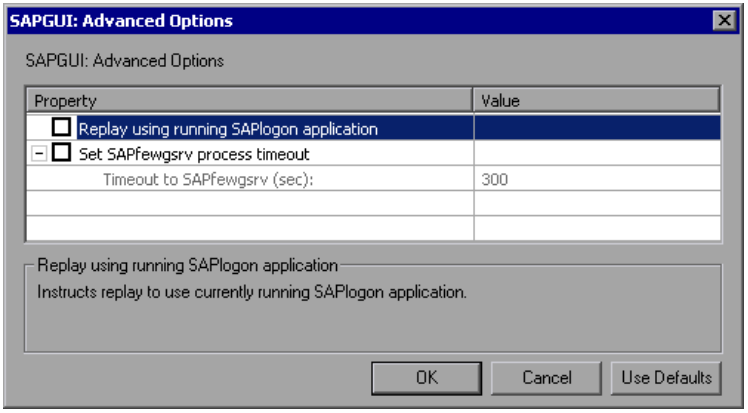
4 In the **Performance** section, select the **Show SAP client during replay** check box to show the SAPGUI user interface during replay.

5 Click **Options** to set a timeout for the **SAPfewgsvr.exe** process.

SAPGUI Advanced Run-Time Settings

Each Vuser invokes a separate **SAPfewgsvr.exe** process during test execution. In some instances, the process stays active even after the replay session has ended. You can check the Windows Task Manager to see if the process is still active.

The Advanced SAPGUI settings let you set a timeout for this application. When the timeout is reached, VuGen closes any **SAPfewgsvr** processes not previously terminated.



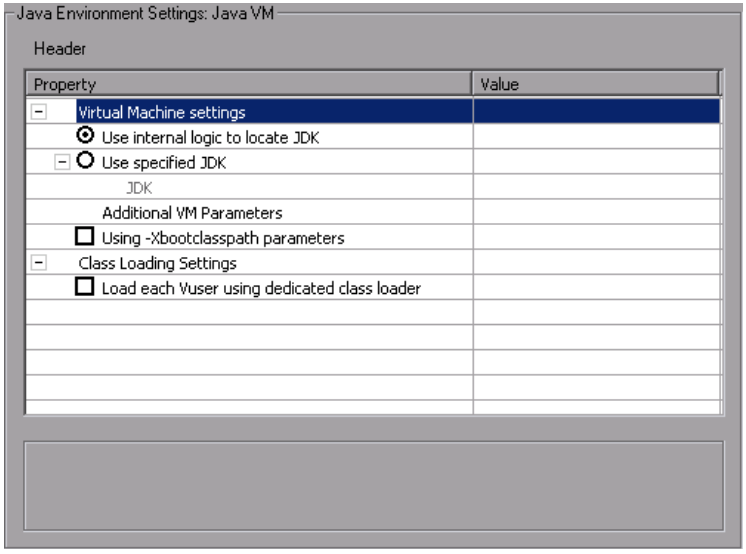
- **Replay using running SAPlogon application**. Instructs the Vusers to use the SAPlogon application that is currently running for replay.
- **Set SAPfewgsvr application timeout**. Allows you to modify the **SAPfewgsvr.exe** process timeout.
- **Timeout to SAPfewgsvr.** The **SAPfewgsvr.exe** process timeout in seconds. The default is 300 seconds.

Java and EJB Run-Time Settings

You set the Java related run-time settings through the **Java VM** options in the Run-Time Settings dialog box.

Specifying the JVM Run-Time Settings

In the Java VM section, you provide information about the Java virtual machine settings.



The following settings are available:

- **Virtual Machine settings**
  - **Use internal logic to locate JDK.** Search the PATH, registry, and Windows folder for the JDK to use during replay.
  - **Use specified JDK.** Use the JDK specified below during replay.
- **Additional VM Parameters.** Enter any optional parameters used by the virtual machine.
  - **Using Xbootclasspath parameters.** Replays the script with the Xbootclasspath /p option.
- **Class Loading Settings**

➤ **Load each Vuser using dedicated class loader.** Load each Vuser using a dedicated class loader. This will allow you to use a unique namespace for each Vuser and manage their resources separately.

To set the Java VM run-time settings:

1 Select **Vuser** > **Run-Time Settings** and select the **Java Environment Settings:Java VM** node in the Run-Time Settings tree.

2 Select the desired **Virtual Machine settings** indicating the JDK to use for the replay.

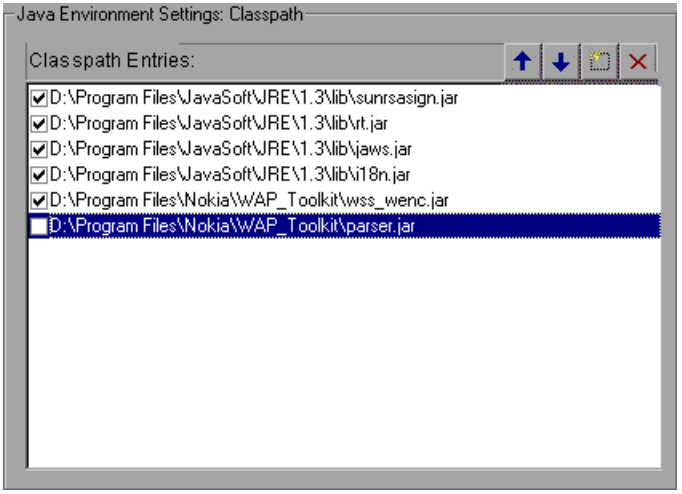
3 To replay with the **-Xbootclasspath/p** option, select the **Using Xbootclasspath parameters** option.

4 Click **OK**.

Setting the Run-Time Classpath Options

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper replay.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.



To set the Classpath run-time settings:

1 Open the Run-Time settings (F4). Select the **Java Environment Settings:Classpath** node in the Run-Time settings tree.

2 Add a classpath to the list:

Click the **Add Classpath** button. VuGen adds a new line to the classpath list.

Type in the path and name of the **jar**, **zip** or other archive file for your class. Alternatively, click the **Browse** button to the right of the field, and locate the desired file. VuGen adds the new location to the classpath list, with an enabled status.

3 To permanently remove a classpath entry, select it and click the Delete button.

4 To disable a classpath entry for a specific test, clear the check box to the left of the entry.

5 To move a classpath entry down in the list, select it and click the Down arrow.

6 To move a classpath entry up within the list, select it and click the Up arrow.

7 Click **OK** to close the dialog box.

RTE Run-Time Settings

You set the Terminal Emulator related run-time settings through the **RTE** node in the Run-Time Settings dialog box.

Modifying Connection Attempts

The **TE_connect** function is generated by VuGen when you record a connection to a host. When you replay an RTE Vuser script, the **TE_connect** function connects the terminal emulator to the specified host. If the first attempt to connect is not successful, the Vuser retries a number of times to connect successfully. Details of each connection are recorded in the report file **output.txt**.

To set the maximum number of times that a Vuser will try to connect, enter a number in the **Maximum number of connection attempts** box in the RTE Run-Time settings.

By default, a Vuser will try to connect 5 times.

For more information about the **TE_connect** function, see the *Online Function Reference* (**Help** > **Function Reference**).

Specifying an Original Device Name

In certain environments, each session (Vuser) requires a unique device name. The **TE_connect** function generates a unique 8-character device name for each Vuser, and connects using this name. To connect using the device name (that is contained within the com_string parameter of the **TE_connect** function), select the **Use original device name** option in the RTE Run-Time settings.

Note: The original device name setting applies to IBM block-mode terminals only.

By default, Vusers use original device names to connect.

For details about the **TE_connect** function, see the *Online Function Reference* (**Help** > **Function Reference**).

Setting the Typing Delay

The delay setting determines how Vusers execute **TE_type** functions.

To specify the amount of time that a Vuser waits before entering the first character in a string, enter a value in the First key box, in milliseconds.

To specify the amount of time that a Vuser waits between submitting successive characters, enter a value in the Subsequent keys box, in milliseconds.

If you enter zero for both the first key and the subsequent key delays, the Vuser will send characters as a single string, with no delay between characters.

You can use the **TE_typing_style** function to override the Delay settings for a portion of a Vuser script.

For details about the **TE_type** and **TE_typing_style** functions, see the *Online Function Reference* (**Help** > **Function Reference**).

Configuring the X-System Synchronization

RTE Vuser scripts use the **TE_wait_sync** function for synchronization. You can set a timeout value and a stable-time value that VuGen applies to all **TE_wait_sync** functions. For details about the **TE_wait_sync** function, see the *Online Function Reference* (**Help** > **Function Reference**).

Timeout

When you replay a **TE_wait_sync** function, if the system does not stabilize before the synchronization timeout expires, the **TE_wait_sync** function returns an error code. To set the synchronization timeout, enter a value (in seconds) in the Timeout section of the RTE Run-Time settings.

The default timeout value is 60 seconds.

Stable Time

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X-SYSTEM mode. After the terminal returns from the X-SYSTEM mode, the Vuser still monitors the system for a short time. This makes sure that the terminal has become stable, that is, that the system has not returned to the X-SYSTEM mode. Only then does the **TE_wait_sync** function terminate.

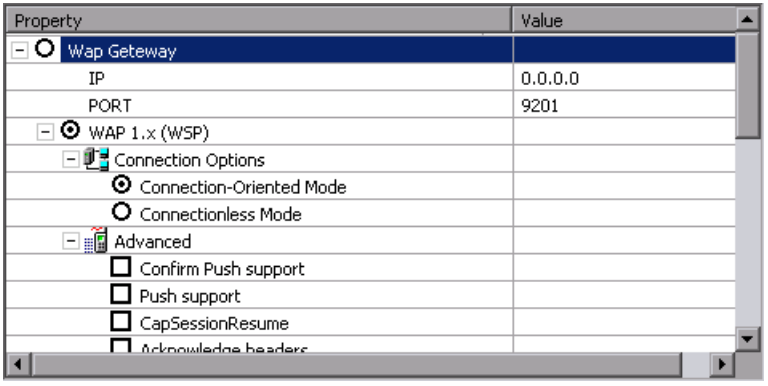
To set the time that a Vuser continues to monitor the system after the system has returned from the X-SYSTEM mode, enter a value (in milliseconds) in the Stable time box of the RTE Run-Time settings.

The default stable time is 1000 milliseconds.

WAP Run-Time Settings

Configuring Gateway Options

You use the **WAP:Gateway** node in the Run-Time Settings tree to set the Gateway settings.



Connection Options

The connection options specify the method that the Vuser uses to connect to the WAP gateway.

WAP Gateway. Run the Vusers accessing a Web server via a WAP Gateway.

HTTP Direct. Run the Vusers run in HTTP mode, accessing a Web server directly.

Note: If you select the HTTP Direct connection mode, the remaining WAP Gateway options are not applicable.

Gateway Settings

If the Vusers connect through a gateway, the IP, Port, and WAP Versions options specify the Gateway connection.

IP. Specify the IP address of the gateway.

Port. Specify the port of the gateway. When running your Vusers through a WAP gateway, VuGen automatically sets default port numbers, depending on the selected mode. However, you can customize the settings and specify a custom IP address and port for the gateway.

Wap version. Select the appropriate WAP version, **1.x (WSP)** or **2.0 (HTTP proxy)**. If you recorded in WAP 1.x (WSP), you can run the Vuser in either 1.x (WSP), or 2.0 (HTTP proxy) mode. If you recorded in WAP 2.0 (HTTP proxy), then you can only run the Vuser in the same mode.

If you are running the script in WAP 1.x (WSP), you can specify several connection and advanced options.

Gateway Connection Mode

The connection mode settings apply to WAP version 1.x (WSP) connections.

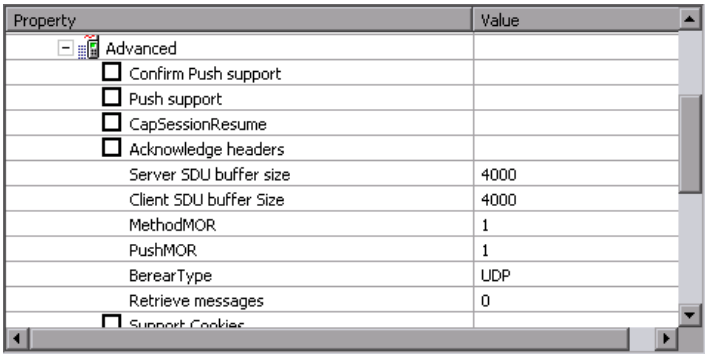
Connection-oriented Mode. Set the connection mode for the WSP session to Connection-Oriented.

Connectionless Mode. Set the connection mode for the WSP session to Connectionless.

Enable security. Enable a secure connection to the WAP gateway.

Advanced Gateway Options

Expand the **Advanced** option in the Gateway node to configure the WAP Capabilities and other advanced gateway options.



- **Confirm Push support.** In CO mode, if a push message is received, this option instructs the Vuser to confirm the receipt of the message (disabled by default). For more information, see "Push Support" on page 1048.
- **Push support.** Enables push type messages across the gateway (disabled by default).
- **CAPSessionResume.** Enables requests for session suspend or resume.
- **Acknowledge headers.** Returns standard headers that provide information to the gateway (disabled by default).
  - **Server SDU buffer size.** The largest transaction service data unit that may be sent to the server during the session (4000 by default).
  - **Client SDU buffer size.** The largest transaction service data unit that may be sent to the client during the session (4000 by default).
  - **MethodMOR.** The number of outstanding methods that can occur simultaneously.
  - **PushMOR.** The number of outstanding push transactions that can occur simultaneously.
  - **BearerType.** The type of bearer used as the underlying transport.

- **Retrieve messages.** When a push messages is received, this option instructs the Vuser to retrieve the message data from the URL indicated in the push message (disabled by default).

- **Support Cookies.** Provide support for saving and retrieving cookies (disabled by default).
- **WTP Segmentation and Reassembly.** Enables segmentation and reassembly (SAR) in WTP, Wireless Transport Protocol. (True by default).
  - **WTP Retransmission Time.** The time in seconds that the WTP layer waits before resending the PDU if it did not receive a response. (5000 by default).
- **WTLS Abbreviated Handshake.** Use an abbreviated handshake instead of a full one, when receiving a redirect message. (False by default).
- **WTLS Deffie Hellman.** Use the Deffie Hellman encryption scheme for WTLS (Wireless Transport Layer Security) instead of the default scheme, RSA. (False by default).
  - **WTLS Deffie Hellman identifier.** An identifier for the Deffie Hellman encryption scheme. This identifier is required for the abbreviated handshake with the Operwave gateway that uses the Deffie Hellman encryption scheme.
  - **Network MTU Size.** the maximum size in bytes, of the network packet. (4096 by default).

Setting the Gateway Options

The following section describes the procedure for setting the WAP Gateway options.

To set the WAP gateway options:

1. Click the **Run-Time Settings** button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **WAP:Gateway** node.
2. To replay the script in WSP mode (not HTTP), select **WAP Gateway**.
3. Specify an IP address and port for the gateway. You can also use the default port indicated by VuGen.
4. Select a WAP version: **WAP 1.x (WSP)** or **WAP 2.0 (HTTP)**.

5 For WAP 1.x (WSP), select a connection mode—**Connection-oriented** or **Connectionless**. To indicate a secure connection mode, select the **Enable Security** option.

6 For WAP 1.x (WSP), expand the **Advanced** node to set the client capabilities and other advanced gateway options. For more information about the Advanced options, see above.

Configuring Radius Connection Data

RADIUS (Remote Authentication Dial-In User Service) is a client/server protocol and software that enables remote access servers to communicate with a central server to authenticate dial-in users and authorize their access to the requested system or service.

RADIUS allows a company to maintain user profiles in a central database that all remote servers can share. It provides better security, allowing a company to set up a policy that can be applied at a single administered network point. Using a central service makes it easier to track usage for billing and store network statistics.

RADIUS has two sub-protocols:

- **Authentication.** Authorizes and controls user access.
- **Accounting.** Tracks usage for billing and for keeping network statistics.

In VuGen, the RADIUS protocol is only supported for WSP replay for both Radius sub-protocols—authentication and accounting.

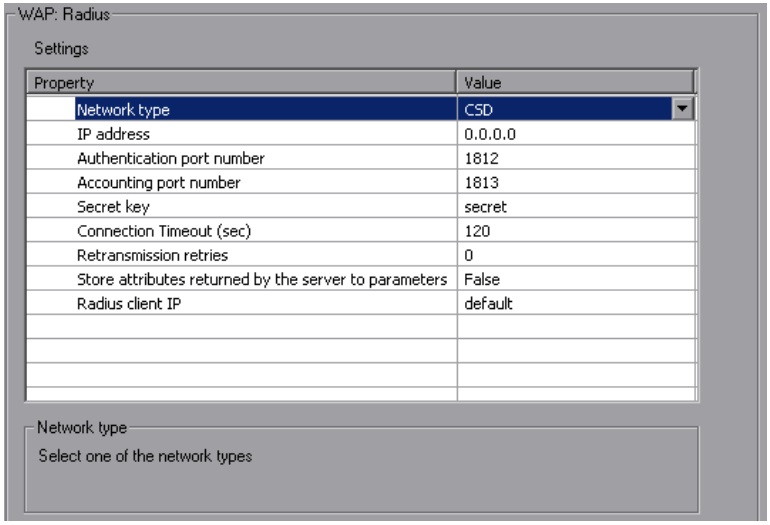
You supply the dial-in information in the Run-Time Settings' Radius node:

Property	Value
Network Type	Accounting network type: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).
IP Address	IP address of the Radius server.
Authentication port number	Authentication port of the Radius server.
Accounting port number	Accounting port of the Radius server.

Secret Key	The secret key of the Radius server.
Connection Timeout (sec)	The time in seconds to wait for the Radius server to respond. The default is 120 seconds.
Retransmission retries	The number of times to retry after a failed transmission. The default is 0.
Store attributes returned by the server to parameters	Allow Vusers to save attributes returned by the server as parameters, which can be used at a later time. The default is **False**.
Radius client IP	Radius packets source IP, usually used to differentiate between packets transmitted on different NIC cards on a single Load Generator machine.

To set the WAP Radius options:

1 Click the **Run-Time Settings** button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Click the **Radius** node.



2 Select an accounting **Network type**: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).

3 Enter the **IP address** of the Radius server in dot form.

4 Enter the **Authentication Port number** and **Accounting Port number** of the Radius server.

5 Type in the **Secret key** for Radius or Accounting Authentication.

6 Enter a **Connection Timeout** value.

7 Specify the number of **Retransmission retries**.

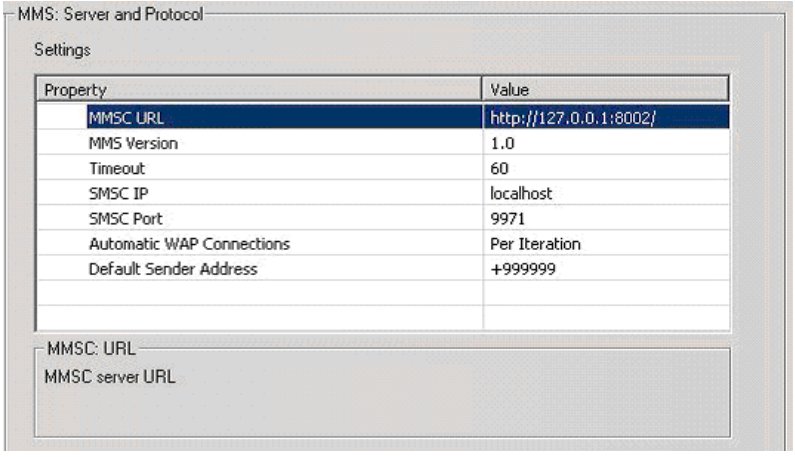
8 Specify whether you want VuGen to **store attributes returned by the server to parameters**.

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

MMS Run-Time Settings

Before running your script, you can set the run-time settings to allow the script to accurately emulate a real user.

The following section describes the run-time settings specific to MMS (Multimedia Messaging Service) Vusers. These run-time setting allow you to configure the server and protocol settings.



You can set the following options:

➤ **MMSC URL.** The URL of the MMSC (Multimedia Messaging Center) server.

- **MMS Version.** The version of the MMS protocol used by the script.
- **Timeout (seconds).** The time that the server waits for incoming messages. The default value is 60 seconds.
- **SMSC IP.** The IP address of the SMSC server used for sending MMS notifications over SMPP.
- **SMSC Port.** The IP port of the SMSC server used for sending MMS notifications over SMPP.
- **Automatic WAP Connections.** Defines when to connect and disconnect from a WAP gateway. This setting is only relevant when a WAP gateway is used. The possible values are:
  - **Per Iteration.** Connect at the beginning of each iteration and disconnect at the end of each iteration. (default)
  - **Per Send or Receive.** Connect and disconnect at the beginning and end of each message.
  - **None.** Do not use automatic WAP connections.
- **Default Sender address.** The default address sent in the Sender header. The default is +999999.

To set the MMS Server and Protocol settings:

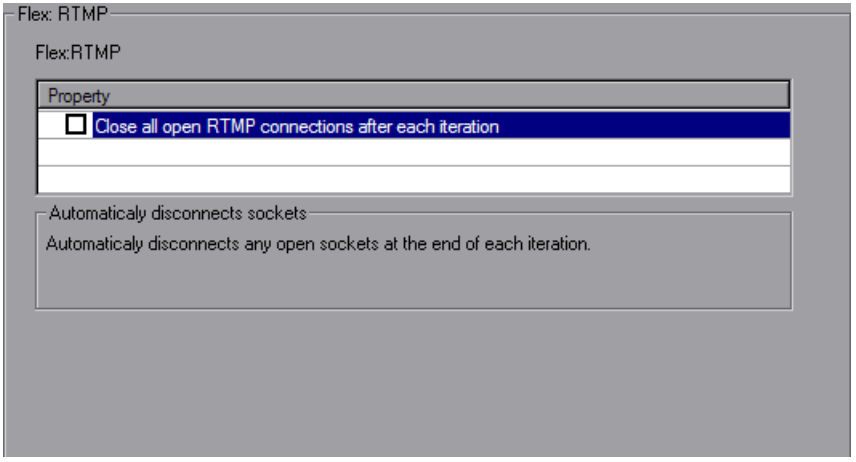
1. Open the Run-Time Settings dialog box. Select **Vuser** > **Run-Time Settings** or click the **Run-Time Settings** button on the VuGen toolbar.
2. Select the **MMS:Server and Protocol** node from the Run-Time settings tree.
3. Select the desired values as explained above.
4. Select **General:Miscellaneous** from the Run-Time settings tree.
5. Under Multithreading, select **Run Vuser as a process.**

6 Click **OK** to accept the settings and run the script.

Flex Run-Time Settings

Before running your script, you can set the run-time settings to allow the script to accurately emulate a real user.

The following section describes the run-time settings specific to Flex Vusers.



You can set the following option:

➤ **Close all open RTMP connections after each iteration.** Automatically disconnects any open RTMP connections at the end of each iteration.

Part 6

Information for Advanced Users

82

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.

This chapter includes:

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 Chapter 37, "Programming a Script in the VuGen Editor."

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 6, "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). Select **File > New** and click the Projects tab.
2. In the Wizard, select *empty dll*.
3. Add the following files to the project:
   - A new *cpp* file with 3 exported function: *init*, *run*, *end* (the names may be customized).

➤ The library file lrun50.lib (located in the <lr installation dir>/lib).

4 In the project settings change the following:

➤ Select the C/C++ tab and select **Code generation** (Category) > **Use Run Time library** (List). Change it to: **Multithreaded dll.**

➤ Select the C/C++ tab and select **Preprocessor** (Category) > **Preprocessor definitions** (edit field) Remove _DEBUG.

5 Add code from your client application, or program as you normally would.

6 Enhance your script with Vuser API functions. For example, **lr_output_message** to issue messages, **lr_start_transaction** to mark transactions, and so forth. For more information, see the General functions in the *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 1346.

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 Select **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 Select **Vuser** > **Advanced** and enter the DLL name in the Advanced dialog box.

4 Select **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 79, "Configuring Run-Time Settings."

5 Select **Vuser** > **Parameter List** to define parameters for your script. The Parameter dialog boxes are identical to those in VuGen. For more information, see Chapter 70, "Working with VuGen Parameters."

Test the script by running it in standalone mode. Select **Vuser** > **Run Vuser**. The Vuser execution window appears while the script runs.

6 Select **File** > **Exit** to close the configuration utility.

83

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.

This chapter includes:

The following information applies primarily to Web, Web Services, and Wireless Vuser scripts.

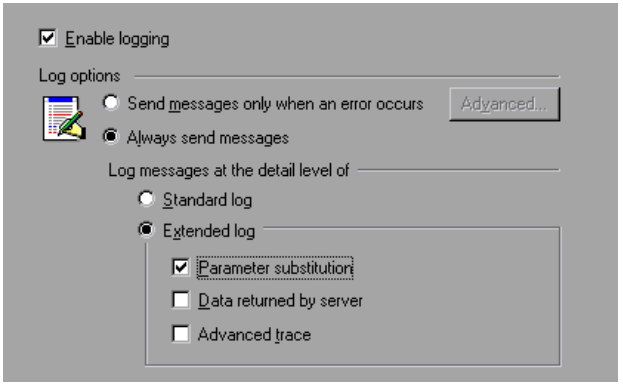
About Programming with the XML API

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 select **Parameter Substitution**. For more information, see Chapter 79, "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.

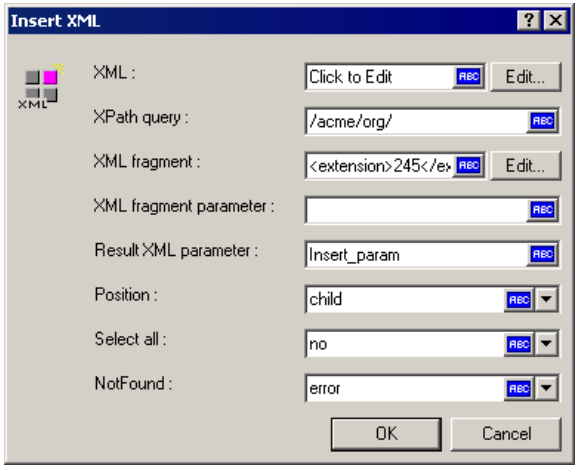
The functions that support result parameters are **lr_xml_insert**, **lr_xml_transform**, **lr_xml_replace**, **lr_xml_delete**, and **lr_xml_set_values**.

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.

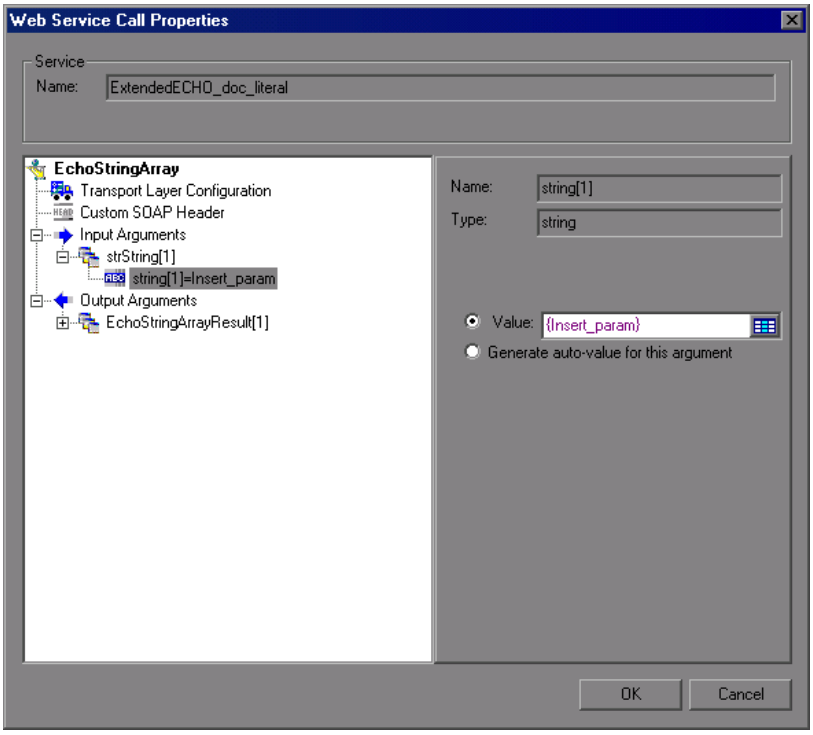
To use the result parameter as input:

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 "Input Argument Values" on page 305.

84

VuGen Debugging Tips

You can use the following integration and configuration tips to help you produce an error-free Vuser script.

This chapter includes:

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 1380.

Question 1: Why does the script fail when it is data driven, while the same values work with the application itself?

Answer: The failure may be a result of trailing spaces in your data values. Even though the data values that you type directly into the GUI are probably truncated, you should manually eliminate them from your data file. Tab-delimited files can hide trailing spaces and therefore obscure problems. In general, comma-delimited files are recommended. You can view the files in Excel to see if things are correct.

Question 2: Why does an SQL error of an invalid cursor state occur on the second iteration?

Answer: The **lrd_close_cursor** function may not have been generated or it may be in the *end* section instead of the *action* section. You will need to add a cursor close function or move it from the *end* section to make the script iterate successfully.

Opening a new cursor may be costly in terms of resources. Therefore, we recommend that you only open a cursor once in the *actions* section during the first iteration. You can then add a new parameter that contains the iteration number as a string by using the Iteration Number type. Call this parameter *IterationNum*. Then, inside the *actions* section replace a call to open a new cursor 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 select the wrong value when correlating the query.

Also verify that there are no problems during multi-user replay. In certain instances, only one user will successfully perform an UPDATE. This occurs with Siebel, where it is necessary to manually write a loop to overcome the problem.

Question 7: The database does not appear to be modified after replaying a script which should have modified the database.

Answer: Through the user application's UI, check if the updated values appear when trying to see the current data accessible to the application. If the values have not been updated, you need to determine they were not changed. Possibly, an UPDATE statement changed one or more rows when the application was recorded, and did not change any during replay.

Check these items:

- **Verify statement.** If there is a WHERE clause in the UPDATE statement, verify that it is correct.
- **Check for correlations.** Record the application twice and compare the UPDATE statements from each of the recordings to make sure that the necessary correlations were performed.
- **Check the total number of rows.** Check the number of rows that were changed after the UPDATE. For Oracle, this information is stored in the fourth parameter of **lrd_exec**. For ODBC, use **lrd_row_count** to determine the number of rows updated. You can also add code to your script that prints the number of rows that were updated. If this value is 0, the UPDATE failed to modify the database.
- **Check the SET clause.** Check the SET clause of the UPDATE statement. Make sure that you correlated any necessary values here instead of hard-coding them. You can see this by comparing two recordings of the UPDATE.

It certain cases, the UPDATE works when replaying one Vuser, but not for multiple Vusers. The UPDATE of one Vuser might interfere with that of another. Parameterize each Vuser so that each one uses different values during the UPDATE, unless you want each vuser to update with the same values. In this case try adding retry logic to perform the UPDATE a second time.

Question 8: How do I avoid the unique column name error when replaying a statement recorded with an Oracle Application. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "
        "MTL_UNITS_OF_MEASURE "
        "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
        "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

The following error message was issued:

```
"lrdo.c/fjParse:  "oparse" ERROR return-code=960, oerhms=ORA-00960:
ambiguous column naming in select list".
```

Answer: Change the statement by adding an alias to at least one of the non-unique columns, thereby mapping it to a new unique name. For example:

```
lrd_stmt(Csr9,"SELECT UOM_CODE,UOM_CODE second, DESCRIPTION
FROM"
        "MTL_UNITS_OF_MEASURE "
        "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
        "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

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
```

85

Advanced Topics

The following advanced information can assist you in determining the replay issues and debugging your Vuser script.

This chapter includes:

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 1392.

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 1394 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 7

Appendixes

86

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

This chapter includes:

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.

87

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.

This chapter includes:

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:

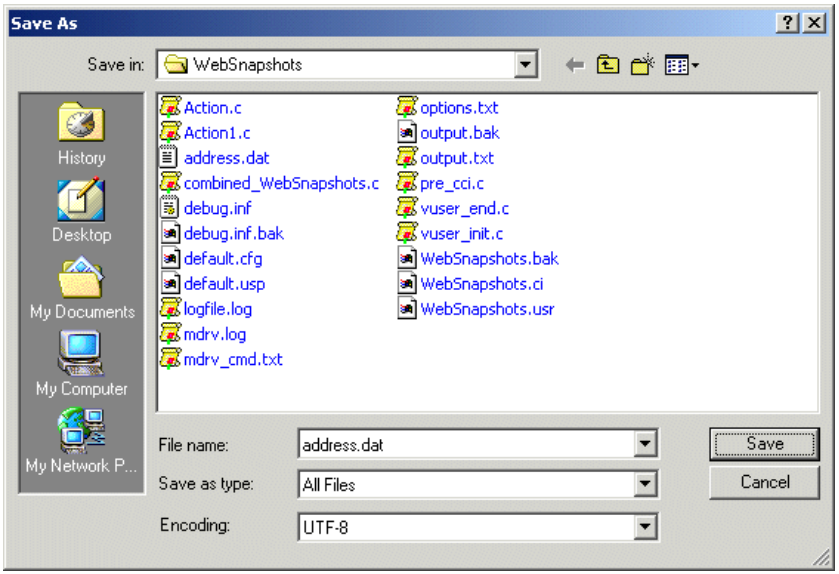
1 Select **Vuser** > **Parameter List** and view the parameter properties.

2 In the right pane, locate the parameter file in the **File path** box.

3 With the parameter table in view, click **Edit in Notepad**. Notepad opens with the parameter file in csv format.

4 In the **Save as type** box, select *All Files*.

In the **Encoding** box, select *UTF-8* type encoding.



5 Click **Save**. Notepad asks you to confirm the overwriting of the existing parameter file. Click **Yes**.

VuGen now recognizes the parameter file as UTF-8 text, although it still displays it in regular characters.

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

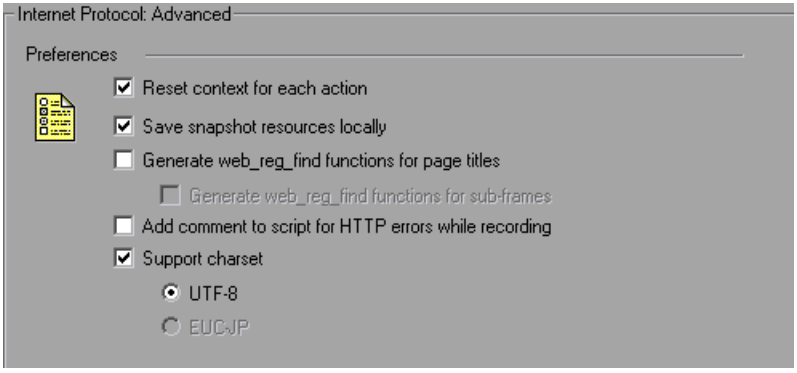
In order to be recognized as a non-English Web page, the page must indicate the charset in the HTTP header or in the HTML meta tag. Otherwise, VuGen will not detect the EUC-JP encoding and the Web site will not be recorded properly. To instruct VuGen to record non-English requests as **EUC-JP** or **UT-8**, 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**. Select 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 "Web, Wireless, and Oracle NCA Recording Options" on page 1168.

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, select **Tools** > **Internet Options and** click the **Advanced** tab. Clear the **Always Send URLs as a UTF-8** option in the Browsing section.

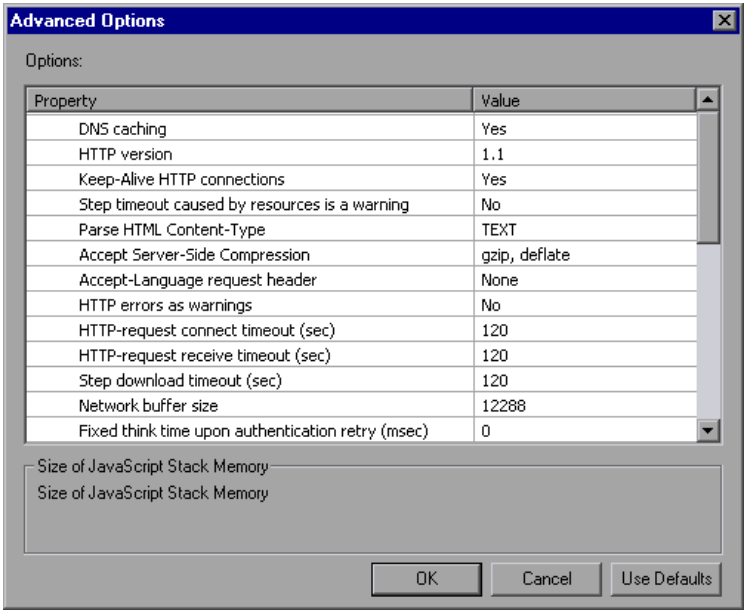
For more information about **web_sjis_to_euc_param**, see the *Online Function Reference*.

Specifying a Language for the Accept-Language Header

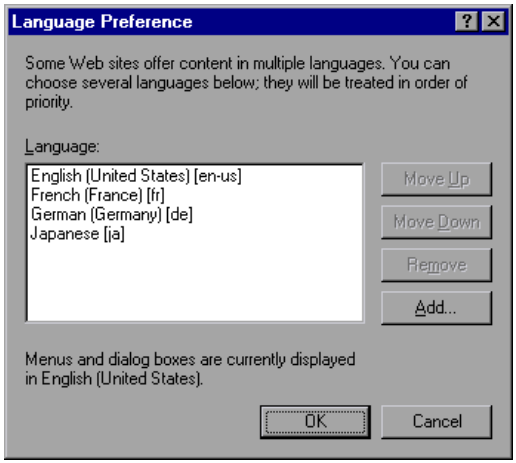
Before running a Web script, you can set the page's request header to match your current language. In the Internet Protocol Run-Time settings, you set the language of the *Accept-Language* request header. This header provides the server with a list of all of the accepted languages.

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 "Additional Options for Internet Preferences" on page 1290.

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.

88

Programming Scripts on UNIX Platforms

Vusers on UNIX platforms can create scripts through programming. To create a script through programming, you use a template.

This chapter includes:

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 6, "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:

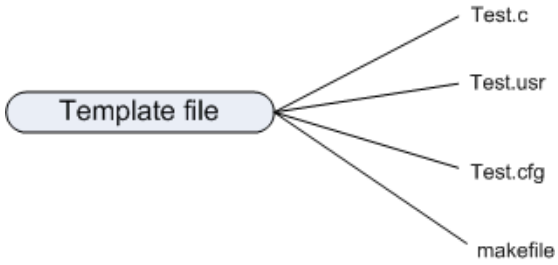
```
mkdbtest name
```

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 79, "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, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

89

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

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

T

U

V

W

X

Z