

HP Virtual User Generator

for the Windows operating system

Software Version: 11.00

User Guide

Document Release Date: September 2010

Software Release Date: September 2010



Legal Notices

Warranty

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

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Copyright Notices

© Copyright 1993 - 2010 Hewlett-Packard Development Company, L.P.

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

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

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

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

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

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

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

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

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

Support

Visit the HP Software Support web site at:

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

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

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

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

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

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

To find more information about access levels, go to:

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

Table of Contents

Using This Documentation Library	19
Welcome to LoadRunner VuGen	19
VuGen	19
How This Guide Is Organized	20
Who Should Read This Guide	20
Accessing the Home Page.....	20
Documentation Library Guides	21
22	
Searching and Navigating the Documentation Library	23
Topic Types	24
Additional Online Resources	26
Documentation Updates	27

PART I: WORKING WITH VUGENSERVICE TEST 29

Chapter 1: Overview	31
VuGen Overview	33
Vuser Overview.....	34
Task Pane Overview.....	36
Script Sections	37
HP LoadRunner Licensing.....	38
VuGen Code Overview.....	39
VuGen Code Tools	43
HP Service Test Features	47
Multiple Protocol Scripts.....	49
Online Resources	50
How to Create a Vuser Script - Workflow	52
How to Create a Business Process Report.....	54
How to Compare Scripts Side by Side	55
Vuser Types.....	57
Keyboard Shortcuts	62
Main User Interface	64

Chapter 2: Protocol Advisor	109
Protocol Advisor Overview.....	110
How to use the Protocol Advisor.....	111
Protocol Advisor User Interface.....	115
Chapter 3: Recording	119
Providing Authentication Information.....	121
Script Directory Files	123
Vuser Script Templates	124
Script Sections	125
How to Create or Open a Vuser Script	127
How to Work with .zip Files.....	129
How to Import Code from a Script	130
How to Record a Vuser Script.....	131
How to Regenerate a Vuser Script	133
How to Create and Open Vuser Script Templates	134
Files Generated During Recording	136
Recording User Interface	138
Chapter 4: Replaying and Debugging Vuser Scripts	147
Replay Overview.....	149
Debugging Features	149
Additional Debugging Information	151
Debugging Features for Web Vusers.....	153
How to Replay a Vuser Script	154
How to Run a Vuser Script from a Command Prompt.....	155
How to Run a Vuser Script from a UNIX command line.....	156
How to Debug Scripts with Breakpoints	159
How to Use Bookmarks	161
Files Generated During Replay	163
Replay User Interface.....	165

Chapter 5: Preparing Scripts for Load Testing	169
Password Encoding.....	171
Encrypting Text.....	171
Transaction Overview.....	172
Rendezvous Points.....	173
Think Time	174
How to Encrypt/Decrypt Text	176
How to Encode a Password.....	177
How to Create a Controller Scenario from VuGen	177
How to Insert Transactions	178
How to Display Transactions	181
How to Prepare a Script for Load Testing.....	182
How to Insert Steps into a Script.....	187
Useful VuGen Functions	190
Preparing Scripts for Load Test User Interface	192
Chapter 6: Viewing Test Results	203
Test Results Overview	204
Customizing the Test Results Display	205
Connecting to Quality Center from the Test Results Window	206
How to Send Custom Information to the Report	207
How to Configure the Appearance of the Test Results Window	207
How to Open the Test Results of a Specific Run	208
How to Find Steps in the Test Results	210
How to Submit Defects to Quality Center	210
Viewing Test Results User Interface	215

Chapter 7: Correlation	229
Correlation Overview	231
Correlation Vs. Parameterization	232
Automatic Correlation	232
Determining Which Values to Correlate	233
Correlating Java Scripts	234
Correlating Java Scripts - Serialization	238
Correlating Winsock Scripts.....	243
Wdiff Utility	244
Modifying Saved Parameters	245
How to Correlate Scripts - Web (Standard).....	246
How to Correlate Scripts - Web (Manual).....	247
How to Correlate Scripts - Oracle NCA	251
How to Correlate Scripts - Database Protocols.....	255
How to Correlate Scripts - Microsoft .NET.....	259
How to Correlate Scripts - Flex and AMF Protocols	261
How to Correlate Scripts - Siebel Protocol	265
How to Correlate Scripts - COM Protocol.....	272
How to Correlate Scripts - Tuxedo Protocol.....	275
How to Correlate Scripts - Winsock (Tree View).....	281
How to Correlate Scripts - Winsock (Script View)	283
How to Search for Values That Require Correlation.....	286
Web_reg_save_param function details.....	290
Correlation Functions - C Vuser Scripts.....	292
Correlation Functions - Java Vuser Scripts.....	293
Correlation Functions - Database Vuser Scripts.....	294
Correlation User Interface	295
Chapter 8: Working with Application Lifecycle Management	299
Managing Scripts Using ALM Overview.....	300
ALM Version Control Overview.....	300
How to Work with Scripts in ALM Projects	302
How to Work with Version Controlled Scripts in ALM Projects	304
How to Save VuGen Scripts to ALM Projects	305
How to View/Modify Previous Versions of a Script	306
ALM User Interface.....	308

Chapter 9: Parameters	315
Parameter Overview	317
Parameter Types	319
Data Assignment Methods for File/Table/XML Parameters.....	322
Tuxedo and PeopleSoft Parameters	328
XML Parameters	329
How to Create a Parameter.....	339
How to Create an XML Parameter from a Web Service Call	341
How to Work with Existing Parameters.....	342
How to Import Parameter Data from a Database.....	343
Parameter User Interface	345
Chapter 10: Recording Options	375
Port Mapping Overview	377
Port Mapping Auto Detection	378
EUC-Encoding (Japanese Windows only).....	379
Script Generation Preference Overview	380
Script Language Options	381
Recording Levels Overview	382
Serialization Overview.....	386
Tips for Working with Event Listening and Recording	387
Example of Click and Script Out of Context Recording.....	389
Protocol Compatibility Table	390
Recording Options User Interface	395
Chapter 11: Run-Time Settings.....	521
Run-Time Settings Overview	522
Error Handling.....	522
Content Checking Overview.....	524
Logging CtLib Server Messages	525
Multithreading	526
Protocol Compatibility Table	528
Run-Time Settings User Interface	534

PART II: PROTOCOLS 613

Chapter 12: AJAX Protocol.....	615
AJAX Protocol Overview	616
AJAX Supported Frameworks	617
AJAX Example Script	617

Chapter 13: Ajax TruClient Protocol	621
Ajax TruClient Protocol Overview	623
Script Detail Levels	623
Private Browsing.....	625
Alternative Steps.....	626
Tips and Tricks	627
How to Record Ajax TruClient Scripts	628
How to Identify Objects	630
How to Enhance Ajax TruClient Scripts	631
How to Insert Custom JavaScript and C Code into Ajax TruClient Scripts	635
How to Modify Steps	637
Step Objects and Properties Reference	638
Ajax TruClient User Interface.....	638
Chapter 14: AMF Protocol	641
AMF Protocol Overview	642
AMF Terms.....	643
AMF Example Script	645
Setting the AMF Recording Mode	646
Chapter 15: Citrix Protocol	653
Citrix Protocol - Overview	655
Citrix Recording Tips	656
Citrix Replaying Tips.....	658
Synchronization	660
Automatic Synchronization	661
Additional Synchronization.....	663
Agent for Citrix Presentation Server Overview	666
How to Configure the Citrix Client and Server	671
How to Synchronize Citrix Scripts Manually	673
How to Install and Uninstall the Citrix Agent	674
Citrix Functions.....	676
Understanding ICA Files	679
Failed Bitmap Synchronization Dialog Box.....	682
Chapter 16: Click and Script Protocols	687
Recording Tips.....	688
Replay Tips	690
Miscellaneous Tips	692
Web (Click and Script) Enhancements	693
Web (Click and Script) API Notes	701

Chapter 17: COM Protocol.....	709
COM Protocol Overview	710
COM Technology Overview.....	710
COM Script Structure	712
COM Sample Scripts.....	715
Selecting COM Objects to Record	721
Chapter 18: Database Protocols	725
Database Protocols Overview	726
VuGen Database Recording Technology.....	727
Database Grids.....	728
Oracle Applications	731
Handling Errors	732
Debugging Database Applications	735
Chapter 19: Enterprise Java Beans (EJB) Protocol	751
EJB Testing Overview	752
EJB Detector Output and Log Files.....	753
How to Create an EJB Vuser Script.....	755
How to Run an EJB Vuser Script.....	759
How to Install and Run the EJB Detector.....	760
How to Set and Verify the Java Environment.....	764
EJB User Interface	768
EJB Vuser Script Examples.....	770
Chapter 20: Flex Protocol.....	777
Flex Overview	778
Code Generation Notifications	779
How to Work with Flex RTMP	780
How to Query an XML Tree	780
How to Handling Externalizable Objects in Flex	783
Flex Functions and Example	787

Chapter 21: Java Protocol	791
Java Protocol Recording Overview	793
Java Vuser Script Overview.....	794
RMI over IIOP Overview	796
Corba Recording Options.....	797
CORBA Application Vendor Classes	798
Recording RMI.....	799
Recording a Jacada Vuser	799
Working with CORBA	801
Working with RMI.....	803
Working with Jacada	805
Java Custom Filters - Overview	806
Java Custom Filters - Determining which Elements to Include	808
How to Record a Java Vuser Script	810
How to Record Java Scripts Using Windows XP and 2000 Server	811
How to Run a Script as Part of a Package.....	812
How to Manually Insert Java Methods	813
How to Manually Configure Script Generation Settings.....	815
How to Create a Custom Java Filter	819
Hook File Structure.....	821
Java Icon Reference List.....	824
Chapter 22: Java Protocol - Manually Programming Scripts.....	827
Manually Programming Java Scripts - Overview.....	828
Java Protocol Programming Tips.....	830
Running Java Vuser Scripts	832
Compiling and Running a Script as Part of a Package.....	833
How to Manually Create a Java Script	835
How to Enhance a Java Script	840
Chapter 23: Java over HTTP Protocol	847
Java over HTTP Protocol Overview	848
How to Record with Java over HTTP.....	849
How to Debug Java over HTTP scripts	851
Chapter 24: LDAP Protocol	855
LDAP Protocol Overview	856
LDAP Protocol Example Script	856
Defining Distinguished Name Entries	858
LDAP Connection Options	860

Chapter 25: Mailing Service Protocols	863
Mailing Service Protocols Overview	864
IMAP Protocol Overview	864
MAPI Protocol Overview	865
POP3 Protocol Overview	867
SMTP Protocol Overview.....	868
Chapter 26: Microsoft .NET Protocol	871
Microsoft .NET Protocol Overview.....	873
Viewing Data Sets and Grids	874
Recording WCF Duplex Communication.....	875
Recording Dual HTTP Bindings.....	881
Asynchronous Calls.....	882
Connection Pooling	884
Debugging Microsoft .NET Scripts	886
Microsoft .NET Filters Overview.....	887
Microsoft .NET Filters - Advanced.....	888
Guidelines for Setting Microsoft .NET Filters.....	890
How to Configure Application Security and Permissions.....	895
Chapter 27: Oracle NCA Protocol	899
Oracle NCA Protocol Overview	900
Oracle NCA Protocol Example Scripts.....	901
Oracle NCA Record and Replay Tips	902
Pragma Mode.....	904
How to Enable the Recording of Objects by Name.....	906
How to Launch Oracle Applications via the Personal Home Page...	909
Chapter 28: RDP Protocol	917
RDP Protocol Overview	919
RDP Recording Tips	920
Working with Clipboard Data.....	921
Image Synchronization Overview	924
Image Synchronization Tips.....	925
Image Synchronization - Shifted Coordinates	927
RDP Agent (Agent for Microsoft Terminal Server) Overview	928
How to Install / Uninstall the RDP Agent.....	932
RDP User Interface.....	934

Chapter 29: RTE Protocol	939
RTE Protocol Overview.....	941
Working with Ericom Terminal Emulation.....	943
Typing Input into a Terminal Emulator.....	945
Generating Unique Device Names.....	948
Setting the Field Demarcation Characters.....	950
Reading Text from the Terminal Screen.....	951
RTE Synchronization Overview.....	953
Synchronizing Block-Mode (IBM) Terminals.....	954
Synchronizing Character-Mode (VT) Terminals.....	958
How to Map Terminal Keys to PC Keyboard Keys.....	964
How to Record RTE Scripts.....	966
Chapter 30: SAP Protocols	971
Selecting a SAP Protocol Type.....	973
SAPGUI Protocol.....	974
SAP Web Protocol.....	978
SAP (Click and Script) Protocol.....	981
Replaying SAPGUI Optional Windows.....	983
How to Configure the SAP Environment.....	985
How to Record SAPGUI Scripts.....	992
How to Replay SAPGUI Script.....	994
How to Run SAPGUI Scripts in a Scenario.....	995
How to Enhance SAPGUI Scripts.....	997
Additional SAP Resources.....	1005
Chapter 31: Siebel Web Protocol	1009
Siebel Web Protocol Overview.....	1010
Siebel Web Recording Options and Run-Time Settings.....	1010
How to Record Transaction Breakdown Information.....	1012
Chapter 32: SilverLight Protocol	1019
Silverlight Protocol Overview.....	1020
How to Import WSDL Files.....	1021
Chapter 33: Tuxedo Protocols	1023
Tuxedo Protocol - Overview.....	1024
Notes About Working with Tuxedo Scripts.....	1025
Defining Environment Settings for Tuxedo Vusers.....	1026
Tuxedo Buffer Data.....	1029

Chapter 34: Web Protocols	1033
Web Protocols Overview	1035
Web Vuser Technology	1036
Web Vuser Types	1037
Support for Push Technology.....	1041
Working with Cache Data.....	1042
Text and Image Verification.....	1043
Data Format Extensions	1046
Web Snapshots	1048
XML Pages	1049
How to Add Text and Image Checks.....	1051
How to Convert Web Vuser Scripts into Java	1052
How to Insert Caching Functions	1054
Data Format Extension List.....	1058
Chapter 35: Web Services - Adding Script Content	1059
Web Service Testing Overview	1061
Adding Web Service Script Content Overview	1061
Special Argument Types	1069
Generating Service Requirements and Tests Overview.....	1073
Server Traffic Scripts Overview.....	1075
How to Add Content.....	1081
How to Assign Values to XML Elements.....	1084
How to Generate a Test Automatically	1086
How to Create a Script by Analyzing Traffic.....	1088
How to Create Business Components in VuGen	1090
How to Create Business Components in Quality Center	1093
Add Script Content User Interface	1095
Aspect Reference.....	1115
Test Generator Wizard User Interface.....	1119
Analyze Traffic User Interface	1125

Chapter 36: Web Services - Managing Services	1131
Managing Services Overview.....	1133
Web Reference Analyzer.....	1139
XML Editing	1141
XML Validation Overview.....	1143
XML Queries.....	1154
How to Add and Manage Services.....	1155
How to Analyze WSDL Dependencies	1157
How to Validate the XML	1158
How to Edit the Schema.....	1161
How to Edit Values in the XML Grid	1163
How to Build an XML Query	1167
Service Management User Interface	1169
SOA Tools User Interface.....	1178
Chapter 37: Web Services - Preparing Scripts for Replay	1187
Preparing for Replay Overview.....	1189
Testing Web Service Transport Layers Overview	1194
JMS Transport Overview.....	1195
Asynchronous Messages Overview.....	1199
Database Integration Overview	1203
Negative Testing Overview.....	1213
Customizing Overview	1215
How to Prepare Scripts for Replay	1225
How to Set up Checkpoints.....	1228
How to Send Messages over JMS	1229
How to Send Messages over HTTP/S	1231
How to Define a Testing Method.....	1234
How to Add a Database Connection.....	1236
How to Create a User Handler.....	1238
How to Customize Configuration Files.....	1243
Tree View Tabs.....	1245
Database Integration User Interface.....	1249

Chapter 38: Web Services - Security	1253
Setting Security Overview	1255
Security Scenarios Overview.....	1263
WCF Scenario Settings.....	1270
Advanced Scenario Setting	1276
Preparing Security Scenarios for Running.....	1282
How to Add Security to a Web Service Script	1286
How to Add SAML Security	1288
How to Customize the Security.....	1289
How to Create and Manage Security Scenarios.....	1294
How to Parameterize Security Elements.....	1297
Set Security User Interface	1299
Security Scenario User Interface	1302
Chapter 39: Web Services - Service Emulation	1317
Emulated Services Overview.....	1318
How to Create an Emulated Service – Workflow	1327
Service Emulation Console User Interface	1331
Chapter 40: Windows Sockets Protocol	1353
Recording Windows Sockets Overview	1354
How to Record a WinSocket Script	1366
How to View and Modify WinSocket Buffers	1369
Data Buffers	1374
Windows Socket User Interface.....	1378
Chapter 41: Wireless Protocols	1385
WAP Protocol Overview	1386
WAP Toolkits.....	1388
Push and Pull Technology.....	1389
VuGen Push Support	1390
MMS (Multimedia Messaging Service) Protocol Overview	1392
How to Run an MMS Scenario in the Controller.....	1394

PART III: ADVANCED TOPICS 1395

Chapter 42: Manually Programming a Script using the VuGen Editor	1397
Manually Programming Scripts - Overview	1398
C Vuser Scripts.....	1399
JavaScript Vusers.....	1401
VBScript Vusers.....	1402
Java Vusers.....	1403
VB Vusers.....	1404

Chapter 43: Creating Scripts with Visual Studio	1407
Creating Vuser Scripts in Visual Studio - Overview	1408
How to Create a Vuser Script with Visual C	1410
How to Create a Vuser Script with Visual Basic.....	1412
How to Configure Run-Time Settings and Parameters	1413
Chapter 44: Language Support.....	1415
Language Support - Overview	1416
Page Request Header Language	1417
How to Convert Encoding Format of a String	1418
How to Convert Encoding Format of Parameter Files	1419
How to Record Web Pages with Foreign Languages	1421
Chapter 45: Advanced.....	1425
Calling External Functions in DLL's	1426
Recording OLE Servers.....	1426
Running a Vuser from the Unix Command Line	1429
Specifying the Vuser Behavior	1431
Command Line Parameters.....	1432
How to Create a New Vuser Type.....	1433
How to Load a DLL locally.....	1438
How to Load a DLL Globally.....	1440
.dat Files.....	1442
Chapter 46: Creating and Running Script in UNIX	1445
Creating and Running Scripts in UNIX - Overview	1446
Programming Vuser Actions	1447
How to Create a Template.....	1449
How to Configure Run-Time Settings Manually.....	1450
How to Define Transaction and Insert Rendezvous Points Manually.....	1454
How to Compile Scripts Manually	1455
Chapter 47: Programming with the XML API	1459
Programming with the XML API - Overview	1460
Using XML Functions.....	1461
Specifying XML Function Parameters	1464
XML Attributes	1466
Structuring XML Scripts	1467
Enhancing a Recorded Session with XML	1469
How to Use Result Parameters.....	1477
Index	1481

Welcome to LoadRunner VuGen

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 Service Management.

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 Service Management helps you optimize the management and availability of business applications and systems in production.

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 Advanced Topics

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.

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.

Documentation Library Guides

The Documentation Library consists of the following guides and references, available online, in PDF format, or both. PDFs can be read and printed using Adobe Reader, which can be downloaded from the Adobe Web site (<http://www.adobe.com>).

Using this Documentation Library explains how to use the Documentation Library and how it is organized.

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. Also 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.
- ▶ **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.

Searching and Navigating the Documentation Library

The following functionality is available from the Documentation Library:

Option	Description
	<p>Search and Navigate. Displays the navigation pane. This button is displayed only when the navigation pane is closed.</p> <p>The navigation pane includes the following tabs:</p> <ul style="list-style-type: none"> ▶ Contents tab. Organizes topics in a hierarchical tree, enabling you to directly navigate to a specific guide or topic. ▶ Index tab. Displays a detailed alphabetical listing of topics, along with the numbers of the pages on which they are mentioned. Double-click an index entry to display the corresponding topic. If your selection occurs in multiple documents, the right pane displays a list of possible locations, enabling you to select a context. ▶ Search tab. Enables you to search for specific topics or keywords. Results are returned in ranked order. You can limit your search to a specific guide by selecting a value from the scope drop-down list. <p>Note: The search looks for each individual word in the phrase and not for full phrases, regardless of whether you use quotations (").</p> ▶ Favorites tab. Enables you to bookmark specific topics for quick reference. <p>The Favorites tab is available only when using the Java implementation of the Help. If your browser does not support Java, the JavaScript implementation is automatically used and the Favorites tab is not displayed.</p>
	<p>Show in Contents. Displays the Contents tab in the navigation pane, and highlights the entry corresponding to the currently displayed page.</p> <p>This button is displayed only when the navigation pane is open.</p>
	<p>Previous and Next. Navigates to the previous or next page in the currently displayed guide.</p>

Option	Description
	Send Documentation Feedback to HP. We welcome your feedback. Use this button in any topic to open an email addressed to us, containing the page reference. Send us your comments, ideas for improvement, and any errors you find.
	Print. Prints the currently displayed page. To print a complete guide, access the printer-friendly link from the Documentation Library Home page.
Back	You can use your browser's Back function to return to the previously displayed page. In most browsers, you can right-click and select Back from the shortcut menu.
Using This Documentation Library	Located on the lower-left corner of each content page. Opens this section.
Glossary	Located on the lower-left corner of each content page. Opens a glossary containing definitions of terms and acronyms.

Topic Types

Note: This section applies to the LoadRunner Controller, VuGen, and Analysis User Guides only.

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

Topic Type	Description	Usage
Concepts 	Background, descriptive, or conceptual information.	Learn general information about what a feature does.
Tasks 	<p>Instructional Tasks. Step-by-step guidance to help you work with the application and accomplish your goals.</p> <p>Task steps can be with or without numbering:</p> <ul style="list-style-type: none"> ▶ Numbered steps. Tasks that are performed by following each step in consecutive order. ▶ Non-numbered steps. A list of self-contained operations that you can perform in any order. 	<ul style="list-style-type: none"> ▶ Learn about the overall workflow of a task. ▶ Follow the steps listed in a numbered task to complete a task. ▶ Perform independent operations by completing steps in a non-numbered task.
	<p>Use-case Scenario Tasks. Examples of how to perform a task for a specific situation.</p>	Learn how a task could be performed in a realistic scenario.

Topic Type	Description	Usage
 <p data-bbox="311 227 504 253">Reference</p>	<p data-bbox="532 227 846 317">General Reference. Detailed lists and explanations of reference-oriented material.</p>	<p data-bbox="908 227 1208 348">Look up a specific piece of reference information relevant to a particular context.</p>
	<p data-bbox="532 378 861 621">User Interface Reference. Specialized reference topics that describe a particular user interface in detail. Selecting Help on this page from the Help menu in the product generally open the user interface topics.</p>	<p data-bbox="908 378 1208 591">Look up specific information about what to enter or how to use one or more specific user interface elements, such as a window, dialog box, or wizard.</p>
 <p data-bbox="311 652 504 704">Troubleshooting and Limitations</p>	<p data-bbox="532 652 875 869">Troubleshooting and Limitations. Specialized reference topics that describe commonly encountered problems and their solutions, and list limitations of a feature or product area.</p>	<p data-bbox="908 652 1208 808">Increase your awareness of important issues before working with a feature, or if you encounter usability problems in the software.</p>

Additional Online 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.

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>).

Welcome to This Guide

Part I

Working with VuGen

1

Overview

This chapter includes:

Concepts

- ▶ VuGen Overview on page 32
- ▶ Vuser Overview on page 32
- ▶ Task Pane Overview on page 34
- ▶ Script Sections on page 34
- ▶ HP LoadRunner Licensing on page 36
- ▶ VuGen Code Overview on page 36
- ▶ VuGen Code Tools on page 39
- ▶ HP Service Test Features on page 43
- ▶ Multiple Protocol Scripts on page 44
- ▶ Online Resources on page 44

Tasks

- ▶ How to Create a Vuser Script - Workflow on page 46
- ▶ How to Create a Business Process Report on page 47
- ▶ How to Compare Scripts Side by Side on page 48

Reference

- ▶ Vuser Types on page 49
- ▶ Keyboard Shortcuts on page 54
- ▶ Main User Interface on page 56

Troubleshooting and Limitations on page 91

Concepts

VuGen Overview

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*.

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.

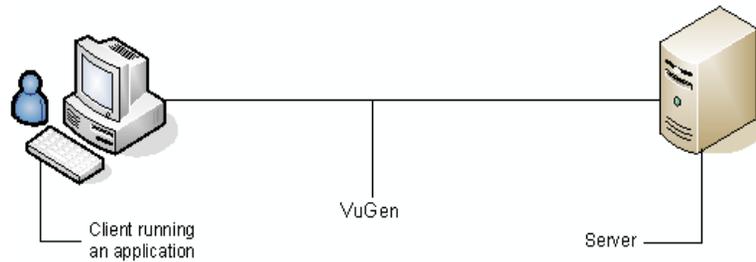
VuGen records Vuser scripts on Windows platforms only. However, a recorded Vuser script can be run on both Windows and UNIX platforms.

Vuser Overview

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 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 HP Business Process Monitor configurations.

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.

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

Task Pane Overview

The Task Pane displays 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.

Script Sections

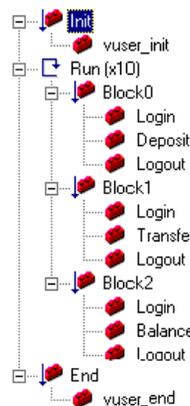
Every Vuser script contains three sections: *vuser_init*, *Run (Actions)*, and *vuser_end*. You can instruct a Vuser to repeat the *Run* section when you run the script. Each repetition is known as an *iteration*.

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

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

- **Action Blocks.** Action blocks are groups of actions within your 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.



- **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.

For user interface details, see "General Run Logic Node" on page 451.

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.

VuGen Code Overview

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

- General Vuser Functions
- Protocol-Specific Vuser Functions
- Standard ANSI C 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 "Preparing Scripts for Load Testing" on page 141 for more information.
- ▶ Send messages to the output, indicating an error or a warning.

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 Javascrpt. For more information, see "General Script Node" on page 353.

All standard conventions apply to the scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other programming languages.

The following segment from a Web Vuser script shows several functions that VuGen recorded and generated in a script:

```
#include "as_web.h"

Action1()
{

    web_add_cookie("nav=140; DOMAIN=dogbert");

    web_url("dogbert",
        "URL=http://dogbert",
        "RecContentType=text/html",
        LAST);

    web_image("Library",
        "Alt=Library",
        LAST);

    web_link("1 Book Search:",
        "Text=1 Book Search:",
        LAST);

    lr_start_transaction("Purchase_Order");
    ...
}
```

For more information about using C functions in your Vuser scripts, see the *Online Function Reference* (Help > Function Reference). For more information about modifying a Java script, see Chapter 22, "Java Protocol - Manually Programming 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.

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 1203.

VuGen Code Tools

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 to search all of the files in the script.

Online Function Reference

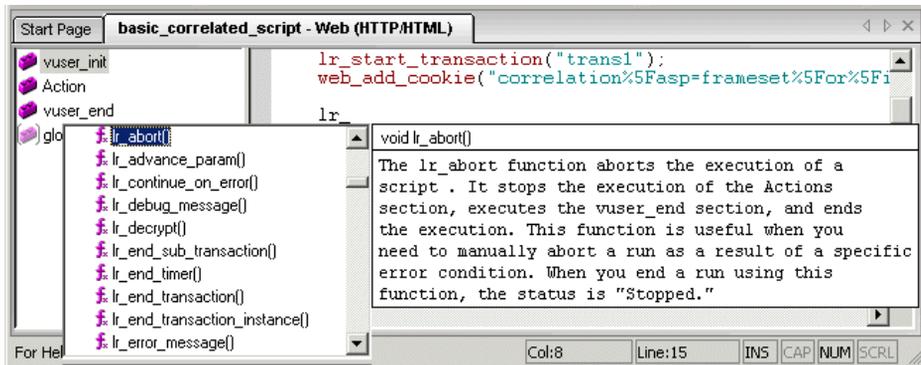
The *Online Function Reference* contains detailed syntax information about all of the VuGen functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

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.

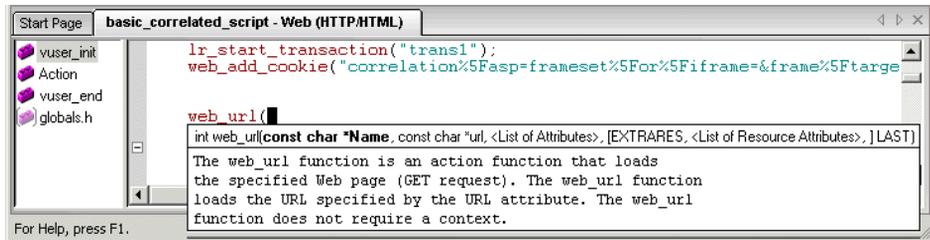


To use one of the displayed functions, select it, or scroll to the desired item and then select it. VuGen inserts the function at the location of the cursor. To close the list box, press the Esc key.

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	lrsiebel.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

HP Service Test Features

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 HP Application Lifecycle Management

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

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

Multiple Protocol Scripts

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 "Java Protocol" on page 671.

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.

Online Resources

VuGen includes the following online tools:

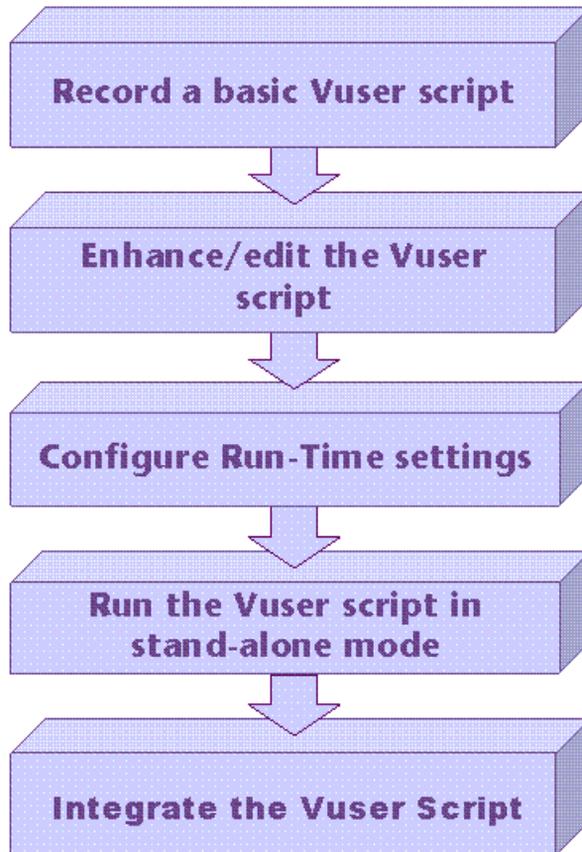
- ▶ **Read Me First.** provides last-minute news and information about VuGen. (Start menu)

- ▶ **Books Online.** displays the complete documentation set in PDF format. Online books can be read and printed using Adobe Acrobat Reader (see www.adobe.com). Check HP's Customer Support Web site for updates to the VuGen online book. (**Help** menu)
- ▶ **Online Function Reference.** gives you online access to all of LoadRunner API functions that you can use when creating Vuser scripts, including examples of how to use them. Check HP's Customer Support Web site for updates to the *Online Function Reference*. (**Help** menu)
- ▶ **Context Sensitive Help.** provides immediate answers to questions that arise as you work with VuGen. It describes dialog boxes, and shows you how to perform standard tasks. To activate this help, click in a window and press F1.
- ▶ **Technical Support Online.** uses your default Web browser to open HP's Customer Support Web site. This site enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is <http://support.hp.com>.
- ▶ **Support Information.** presents the locations of HP's Customer Support Web site and home page, the email address for sending information requests, and a list of HP's offices around the world. (**Help** menu)
- ▶ **HP on the Web.** uses your default Web browser to open HP's home page (<http://www.hp.com>). This site enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more.
- ▶ **LoadRunner Tutorial.** in PDF format, is provided with the downloadable version of LoadRunner. It guides you through the process of creating a load test script for a Web application.

Tasks

How to Create a Vuser Script - Workflow

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



The process of creating a Vuser script is as follows:

- 1** Record a basic script using VuGen. 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 configuration. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Service Management User Guides*.

How to Create a Business Process Report

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.

Note: Business Process Reports are only available for the following protocols: 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.

This task includes the following steps:

- "Create a business process report" on page 48
- "Configure advanced options" on page 48

1 Create a business process report

Select **File > Create Business Process Report** and complete the dialog box. For user interface details, see "Business Process Report Dialog Box" on page 56.

2 Configure advanced options

To modify the advanced report options such as the table of contents, snapshots, and the Microsoft Word template click the **Advanced** button. For user interface details, see "Advanced Dialog Box" on page 58.

How to Compare Scripts Side by Side

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

To compare Vuser scripts:

- 1 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.

Note: You can change the comparison tool from the General Options > Environment Tab. For more information, see "General Options Dialog Box" on page 66.

Reference

Vuser Types

The following table lists the Vuser types, descriptions, and categories:

Protocol	Description	Protocol Category
AJAX (Click and Script)	An acronym for Asynchronous JavaScript and XML. AJAX uses asynchronous HTTP requests, allowing Web pages to request small bits of information instead of whole pages.	E-Business
Ajax TruClient	An advanced protocol for modern JavaScript based applications (including Ajax) emulating user activity within a web browser. Scripts are developed interactively in Mozilla Firefox.	E-Business
Action Message Format (AMF)	Action Message Format, a Macromedia proprietary protocol that allows Flash Remoting binary data to be exchanged between a Flash application and an application server over HTTP.	E-Business
C Vuser	A generic virtual user which uses the standard C library.	Custom
Citrix_ICA	A remote access tool, allowing users to run specific applications on external machines.	Application Deployment Solution
COM/DCOM	Component Object Model (COM) - a technology for developing reusable software components.	Distributed Components
DB2 CLI	The IBM Call Level SQL Interface to the DB2 family of databases.	Client/Server

Protocol	Description	Protocol Category
Domain Name Resolution (DNS)	<p>The DNS protocol is a low-level protocol that allows you to emulate the actions of a user working against a DNS server.</p> <p>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.</p>	Client/Server
EJB Testing	Enterprise Java Beans - an architecture for the development and deployment of Java-server components.	Enterprise Java Beans
Flex	Flex is an application development solution for creating Rich Internet Applications (RIAs) within the enterprise and across the Web.	E-Business
File Transfer Protocol (FTP)	<p>File Transfer Protocol - a system which transfers files from one location to another over a network.</p> <p>The FTP protocol is a low-level protocol that allows you to emulate the actions of a user working against an FTP server.</p>	E-Business
i-mode	NTT DoCoMo's technology for accessing the Internet on a mobile phone system.	
Informix	An IBM Informix database using a standard client/server architecture	Client/Server
Internet Messaging (IMAP)	Internet Message Application - a protocol which enables clients to read email from a mail server.	Mailing Services

Protocol	Description	Protocol Category
Java over HTTP	Designed to record java-based applications and applets. It produces a Java language script using web functions. This protocol is distinguished from other Java protocols in that it can record and replay Java remote calls over HTTP.	
Java Record Replay		
Java template	Java programming language with protocol level support.	Custom
Javascript Vuser	A scripting language used to develop Internet applications.	Custom
Listing Directory Service (LDAP)	An Internet protocol designed to allow email applications to look up contact information from a server.	E-Business
Media Player (MMS)	Streaming data from a media server using Microsoft's MMS protocol. Important: In order to replay Media Player functions, you must place a file called <code>wmload.asf</code> on the Windows Media server machine. The VuGen machine must be able to access using <code>mms://<servername>/wmload.asf</code> . This ASF file can be any media file renamed to <code>wmload.asf</code> .	Streaming
Microsoft .NET	Supports the recording of Microsoft .NET client-server technologies.	Client/Server, Distributed Components, E-Business
Microsoft Remote Desktop Protocol (RDP)	A remote access tool using the Microsoft Remote Desktop Connection to run applications on an external machine.	Application Deployment Solution
MS Exchange (MAPI)	Messaging Application Programming Interface designed to allow applications to send and receive email messages.	Mailing Services

Protocol	Description	Protocol Category
MS SQL Server	Microsoft's SQL Server using the Dblib interface.	Client/Server
Multimedia Messaging Service (MMS)	A messaging service used for sending MMS messages between mobile devices.	Wireless
ODBC	Open Database Connectivity - a protocol providing a common interface for accessing databases.	Client/Server
Oracle (2-Tier)	Oracle database using a standard 2-tier client/server architecture.	Client/Server
Oracle NCA	Oracle 3-tier architecture database consisting of Java client, Web server and database.	ERP/CRM
Oracle Web Applications 11i	The Oracle Applications interface that performs actions over the Web. This Vuser type detects actions on both the Mercury API and Javascript levels.	ERP/CRM
Peoplesoft Enterprise	An Enterprise Resource Planning system based on the PeopleSoft 8 enterprise tools.	ERP/CRM
Peoplesoft-Tuxedo	An Enterprise Resource Planning system based on the Tuxedo Transaction Processing Monitor, including automatic correlation.	ERP/CRM
Post Office Protocol (POP3)	A protocol designed to allow single computers to retrieve email from a mail server	Mailing Services
Real	A protocol used to transfer streaming data from a media server.	Streaming
SAP (Click and Script)	Emulation of communication between a browser and SAP server on a GUI or user-action level.	ERP/CRM

Protocol	Description	Protocol Category
SAPGUI	An Enterprise Resource Planning system to integrate key business and management processes using the SAPGUI client for Windows.	ERP/CRM
SAP - Web	An Enterprise Resource Planning system to integrate key business and management processes using the SAP Portal or Workplace clients.	ERP/CRM
Siebel - Web	A Customer Relationship Management Application.	ERP/CRM
Silverlight	A protocol for Silverlight based applications emulating user activity at the transport level. Allows generating high level scripts by automatically importing and configuring WSDL files used by the application.	E-Business
Simple Mail Protocol (SMTP)	Simple Mail Transfer Protocol - a system for distributing mail to a particular machine.	Mailing Services
Sybase Ctlib	A client/server architecture database called via the Ctlib interface.	Client/Server
Sybase Dblib	A client/server architecture database called via the Dblib interface.	Client/Server
Terminal Emulation (RTE)	Emulation of users who submit input to, and receive output from, character-based applications.	Legacy
Tuxedo	Tuxedo Transaction Processing Monitors.	Middleware
VB Script Vuser	Visual Basic Scripting Edition language - used for programming documents displayed in Web browsers.	Custom
Visual Basic template	Vuser scripts written in Visual Basic language.	Custom

Protocol	Description	Protocol Category
WAP	Wireless Application Protocol - used for Web-based, wireless communication between mobile devices and content providers.	Wireless
Web (Click and Script)	Emulation of communication between a browser and Web server on a GUI or user-action level.	E-Business
Web (HTTP/HTML)	Emulation of communication between a browser and Web server on an HTTP or HTML level.	E-Business
Web Services/ SOA	Web Services are a programmatic interface for applications to communicate with another over the World Wide Web.	E-Business
Windows Sockets	The standard network programming interface for the Windows platform.	Client/Server

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**).

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

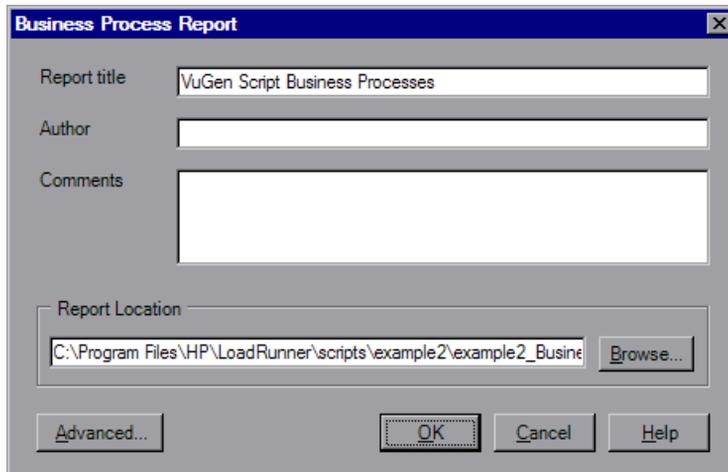
Main User Interface

This section includes:

- ▶ Main VuGen User Interface on page 72
- ▶ Snapshot Pane on page 90
- ▶ Output Window on page 79

Business Process Report Dialog Box

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



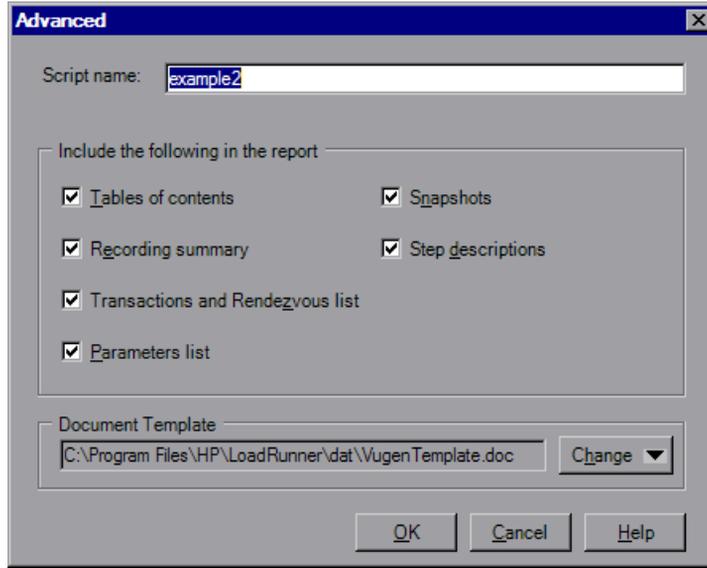
To access	File > Create Business Process Report
Relevant tasks	"How to Create a Business Process Report" on page 47

User interface elements are described below:

UI Elements (A-Z)	Description
	Opens the Advanced dialog box.
Author	Your name.
Comments	Any additional comments you want to appear in the report.
Report location	The location of the report. Default value: script directory.
Report title	The title of the report.

Advanced Dialog Box

This dialog box enables you to specify advanced options when creating a business process report.



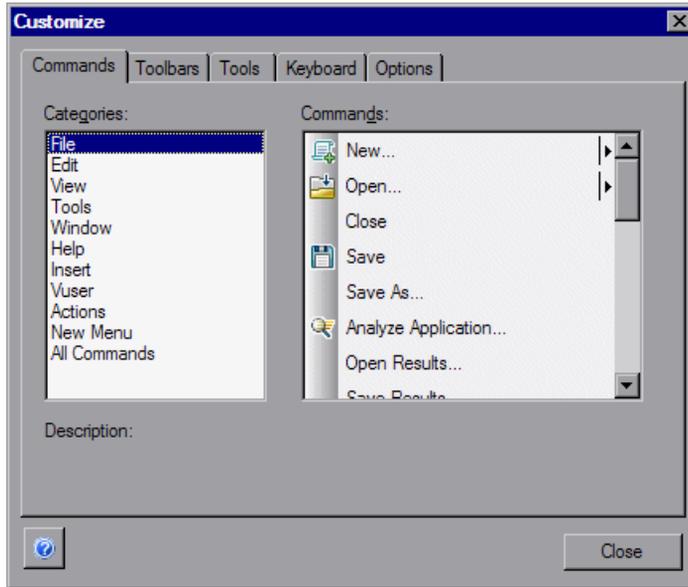
To access	File > Create Business Process Report > Advanced
Relevant tasks	"How to Create a Business Process Report" on page 47

User interface elements are described below:

UI Elements (A-Z)	Description
Document Template	<p>The path and file name of the template to use for the report. The default template is stored in the product's dat folder.</p> <p>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.</p>
Parameter list	<p>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).</p> <p>Default value: enabled.</p>
Recording Summary	<p>A summary of the recording session as it appears when you click the Recording Summary link in the Tasks list.</p> <p>Default value: enabled.</p>
Script name	<p>The .usr file name of the script.</p>
Snapshots	<p>An actual snapshot of the recorded step, adjacent to the step name and description.</p> <p>Note: Oracle NCA and Web Services reports do not include snapshots.</p> <p>Default value: enabled.</p>
Step descriptions	<p>A short description of each step listed in the Tree view.</p> <p>Default value: enabled.</p>
Table of Contents	<p>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.</p> <p>Default value: enabled.</p>
Transactions and Rendezvous list	<p>A comprehensive list of all of the transactions and rendezvous that were defined in the script.</p> <p>Default value: enabled.</p>

Customize Dialog Box

This dialog box enables you to customize the main VuGen toolbars and menus.



To access	Tools > Customize
-----------	-------------------

Commands Tab

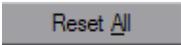
User interface elements are described below:

UI Elements (A-Z)	Description
Categories Pane	A list of categories each containing different commands. Selecting a category to display its commands in the commands pane.
Commands Pane	A list of commands. Click on a command to view its description. To add a command to the toolbar, drag the command from the command pane to the desired location on the toolbar.

Toolbars Tab

User interface elements are described below:

Select the check box for each toolbar you want to display.

UI Elements (A-Z)	Description
	Returns to the default toolbar view settings for all toolbars. VuGen issues a warning before reverting back to the default settings.
	Returns to the default toolbar view settings for toolbar selected in the left pane.
	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.
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.

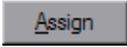
User interface elements are described below:

UI Elements (A-Z)	Description
Arguments	Run-time arguments to be executed with the specified file.
Command	The name of an executable or batch 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.
Initial directory	The initial directory in which to run the custom tool.
Menu contents list	The list of added items. Use the New , Delete , Move Item Up , and Move Item Down buttons to manage this list.

Keyboard Tab

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

User interface elements are described below:

UI Elements (A-Z)	Description
	Assigns the shortcut from the Press New Shortcut Key to this command.
	Removes the selected shortcut from the Current Keys list.
Category	A list of command categories.
Commands	A list of commands.
Current Keys	The list of shortcut keys assigned to the selected command.
Press New Shortcut Key	Put the cursor in this field and press any key. The entry can then be assigned to a command by pressing Assign .

Options Tab

User interface elements are described below:

UI Elements (A-Z)	Description
	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.
Personalized Menus and Toolbars	Menu 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.
Toolbar	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).

Find Dialog Box

This dialog box enables you to find strings in the VuGen code.



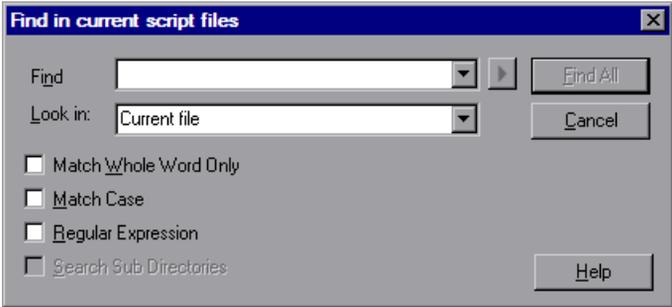
To access	Edit > Find
-----------	-------------

User interface elements are described below:

UI Elements (A-Z)	Description
Find What	Specify the words to search for.
Match Whole Word Only	Searches for occurrences that are whole words, and not part of a larger word.
Match Case	Distinguishes between uppercase and lowercase characters during the search.
Regular Expression	Indicates that the search string is regular expression.
Direction	Select the direction by which you want to search: Up or Down .
	Finds the next occurrence of the text in the Find What box.

Find in Current Script Files Dialog Box

This dialog box enables you to find and replace strings in the VuGen code.



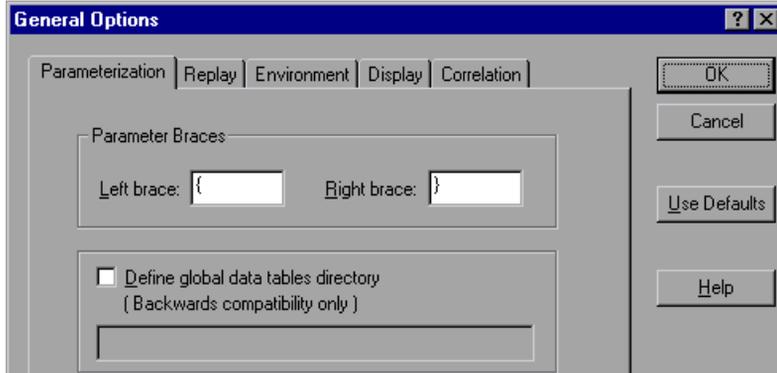
To access	Edit > Find in Current Script Files
-----------	-------------------------------------

User interface elements are described below:

UI Elements (A-Z)	Description
Find What	Specify the words to search for.
Look In	The files in which to search: Current file or all Action pane files , all files that are shown in the file list in the left pane.
Match Whole Word Only	Searches for occurrences that are whole words, and not part of a larger word.
Match Case	Distinguishes between uppercase and lowercase characters during the search.
Regular Expression	Indicates that the search string is regular expression.
Direction	Select the direction by which you want to search: Up or Down .
Search Sub Directories	Perform a search in all of the script's subdirectories.
	Finds all occurrences of the text in the Find box.

General Options Dialog Box

This dialog box enables you to configure a variety of general options including display, parameter, and correlation options.



To access	Tools > General Options
-----------	-------------------------

Parameter Tab

UI Elements (A-Z)	Description
Parameter Braces	<p>When you insert a parameter into a Vuser script, VuGen places parameter braces on either side of the parameter name. 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.</p>
Define global data tables directory	<p>This option is provided only for backward compatibility with earlier versions of VuGen. In 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 this dialog box, you can change the location of the global tables at any time.</p> <p>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 1140.</p> <p>To enable this option, select the Define global data tables directory check box, and specify the directory containing your global data tables.</p> <p>By default, the Define global data tables directory option is disabled.</p>

Replay Tab

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.

UI Elements (A-Z)	Description
After Replay	<p>Instructs VuGen how to proceed after the replay:</p> <ul style="list-style-type: none"> ➤ View before replay. Return to the view you had before replay. Selected by default. ➤ 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.)
Debug	<ul style="list-style-type: none"> ➤ Animate 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. Enabled by default.
Results Directory	<p>Prompt for results directory. Prompts you for a results directory before running a script from VuGen. Disabled by default. If this option is not selected, VuGen automatically names the directory <i>result1</i>. 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.</p>

Environment Tab

UI Elements (A-Z)	Description
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.
Comparison Tool	You can select a comparison tool to be used when comparing two scripts. VuGen comes with a default comparison tool. To view two scripts side by side, select Tools > Compare with Script .
Editor	<p>You can set the editor options to select a font and enable VuGen's Intellisense capabilities which automatically fill in words and function syntax.</p> <ul style="list-style-type: none"> ▶ 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. This option is enabled by default. If you disable it, you can still enable it locally 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 displaying all available matches to the function prefix, along with the function's syntax and description. This option is enabled by default. If you disable it, you can enable it locally by pressing Ctrl+Space or choosing Edit > Complete Word while typing in the editor. ▶ Select Font... . Opens the Select Font dialog box. Select the desired font, style, and size. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.

Display Tab

This tab contains options that only apply to Web vuser scripts. For more information on how to use these options for debugging, see "Debugging Features for Web Vusers" on page 127.

UI Elements (A-Z)	Description
Generate report during script execution	Instructs a Vuser to generate a Results Summary report. This options is enabled by default. You can open the report after script execution by selecting View > Test Results .
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. This option is disabled by default.

Correlation Tab

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 as well as which characters should be treated as delimiters.

UI Elements (A-Z)	Description
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.
Ignore differences shorter than X characters.	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. Default value: 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.

Citrix Display Tab

(Citrix vuser scripts only)

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.

UI Elements (A-Z)	Description
Show client during replay	Displays the Citrix client when replaying the Vuser script.
Show Bitmap Selection popup	Issues 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.

Main VuGen User Interface

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

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

User interface elements are described below:

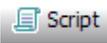
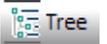
UI Elements	Description
<p>Tree View Tab (tree view only)</p>	<p>Displays the Vuser script in an icon-based format, with each step represented by a different icon.</p> <p>Use the drop-down list above the tree to select a section of the script to display.</p> <p>You can manipulate steps by dragging them to the desired location. You can also add additional steps between existing steps in the tree hierarchy.</p>
<p>Thumbnails Tab (tree view only)</p>	<p>For several Vuser types such as Web, SAPGUI and Citrix, you can view thumbnail representations of the snapshots.</p> <p>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.</p> <p>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.</p>
<p>Action Pane</p>	<p>Divides the script into its major sections. Click on a section to display its contents in the other panes. This pane is displayed by default in the script view. To change the situations in which this pane is displayed, select View > Actions and select the desired option(s).</p>

UI Elements	Description
<p>Script Pane (script view only)</p>	<p>The Script pane 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.</p> <p>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 "VuGen Code Tools" on page 39.</p> <p>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.</p>
<p>Snapshot Pane</p>	<p>A snapshot is a graphical representation of the current step. Depending on the type of protocol, the snapshot displays different types of data about the selected step.</p> <p>For detailed user interface information about the snapshot panes for different protocols, see "Snapshot Pane" on page 90.</p> <p>VuGen captures a base snapshot during recording and another one during replay. You can compare the Record and Replay snapshots to determine the dynamic values that need to be correlated in order to run the script.</p> <p>Each time you replay the script, VuGen saves another Replay snapshot to the script's result directory: Iteration1, Iteration2, and so forth.</p> <p>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.</p> <p>The following toolbar buttons let you show or hide the various snapshot windows.</p>
	<p>View recording snapshots only.</p>

UI Elements	Description
	<p>View both recording and replay snapshots.</p>
	<p>View replay snapshot only.</p>
<p>Output Window</p>	<p>A pane on the bottom of the main window displaying data collected during replay and recording.</p>
<p>Task Pane</p>	<p>Displays 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. For more information, see "Task Pane Overview" on page 34.</p>
<p>Advanced Toolbar</p>	
	<p>Insert New Parameter.</p>
	<p>Insert New Step. Inserts a new step into your script (tree view). For task details, see "How to Insert Steps into a Script" on page 155.</p>
	<p>View Test Results. Opens the Test Results Window. For more information, see "Test Results Window" on page 174.</p>
<p>Debug Tools Toolbar</p>	
	<p>Step.</p>
	<p>Toggle Breakpoint.</p>
	<p>Enable/Disable Breakpoint.</p>
<p>Edit Toolbar</p>	
	<p>Cut.</p>
	<p>Copy.</p>

UI Elements	Description
	Paste.
	Undo.
	Comment Selection.
	Uncomment Selection.
	Increase Indent.
	Decrease Indent.
Record Toolbar	
	Start recording your application. Initializes the recording session.
	Run. Runs/replays the script.
	Stop. Stops a script.
	Pause. Pauses a script while running.
	Compile. Compiles the script.
	Create New Action. Creates a new action section in your script.
	Insert Start Transaction. Inserts a start transaction marker at the specified location.
	Insert End Transaction. Inserts an end transaction marker at the specified location.
	Edit Recording Options. Opens the Recording Options dialog box. For more information see "Recording Options" on page 307.

UI Elements	Description
	<p>Insert Rendezvous. This options is only available in the floating toolbar when recording. For information about rendezvous points, see "Rendezvous Points" on page 144.</p>
	<p>Insert Comment. Inserts a comment into the script. This options is only available in the floating toolbar when recording.</p>
	<p>Insert Text Check. This options is only available in the floating toolbar when recording a Web (HTTP/HTML) or related script. This button inserts a text check into your script during recording. For more information about text checks, see "How to Add Text and Image Checks" on page 896.</p>
	<p>Insert Sync on Bitmap. This options is only available in the floating toolbar when recording Citrix and RDP scripts. It allows you to insert a synchronization step based on a specified area of the snapshot.</p>
	<p>Insert Sync on Text. This options is only available in the floating toolbar when recording Citrix and RDP scripts. It inserts a synchronization step based on a specified selection of text.</p>
	<p>Insert Sync on Image. This options is only available in the floating toolbar when recording an RDP script. It inserts a synchronization step based on a specified image.</p>
<p>Standard Toolbar</p>	
	<p>New. Create a new Vuser script.</p>
	<p>Open. Open an existing Vuser script.</p>
	<p>Save. Save the current script.</p>
	<p>Protocol Advisor. Opens the Protocol Advisor. For more information, see "Protocol Advisor" on page 93.</p>
<p>TreeView Toolbar</p>	

UI Elements	Description
	Properties.
	Insert After.
	Insert Before.
	Delete Step.
Tools Toolbar	
	Create Controller Scenario. Creates a basic controller scenario while remaining in the VuGen application. For details, see "How to Create a Controller Scenario from VuGen" on page 147.
	HP ALM. Opens a connection to HP Application Lifecycle Management, allowing you to save and work with scripts saved in ALM. For more information, see "Working with Application Lifecycle Management" on page 245.
View Toolbar	
	View Tasks. Displays the task pane. For more For more information see the description of the task pane above.
	View Script. Switch to the script view. For more information about the UI elements in the script view see above.
	View Tree. Switch to the tree view. For more information about the UI elements in the tree view see above.
	Show/Hide Output Window. Displays the output window. For more information see the description of the output window above.
Vuser Toolbar	

UI Elements	Description
	Open Parameter List. Opens the Parameter List dialog box. For details, see "Parameter List Dialog Box" on page 301.
	Edit Runtime Settings. Opens the Run-Time Settings dialog box. For more information, see "Run-Time Settings" on page 417.

Output Window

This page dialog box display logs containing information collected during recording and replay phases.

To access	View > Output Window
-----------	----------------------

This window contains the following tabs:

"Output Window - Correlation Results Tab" on page 80

"Output Window - Generation Log Tab" on page 82

"Output Window - Parameters Tab (Winsock only)" on page 84

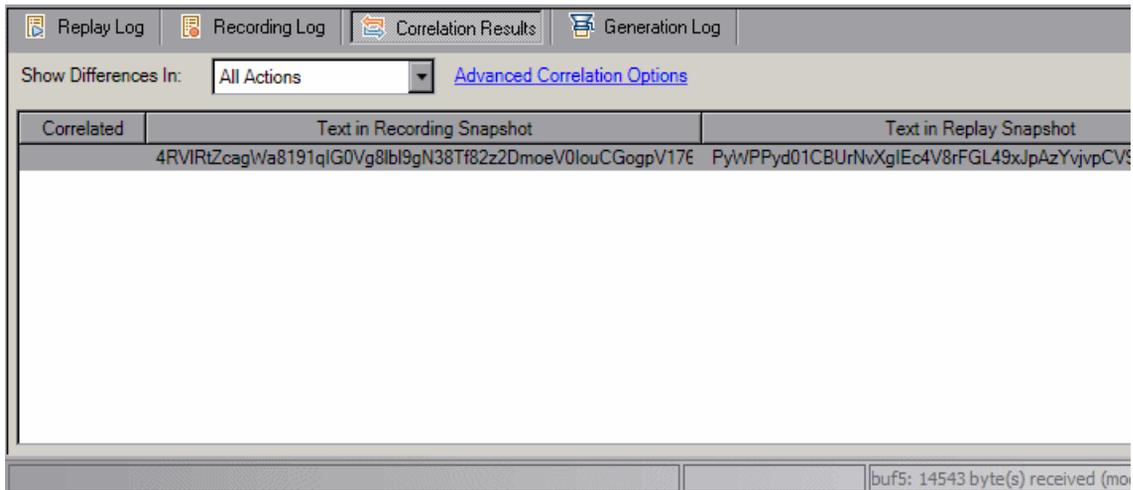
"Output Window - Recording Log Tab" on page 86

"Output Window - Replay Log Tab" on page 87

"Output Window - Run Time Data Tab" on page 88

Output Window - Correlation Results Tab

Displays the differences between the Record and Replay snapshots. Also, allows you to create and manage correlations.



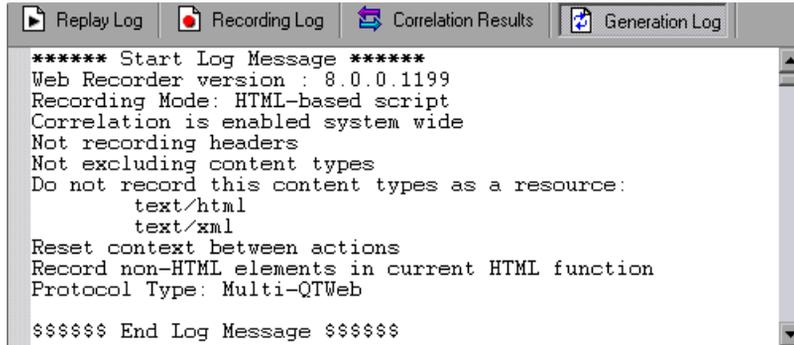
To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

User interface elements are described below:

UI Elements (A-Z)	Description
	<p>Creates a correlation from the selected string. This adds a web_reg_save_param_* function and saves the original value in a comment in the script. Appropriate occurrences of the original value in web steps are replaced with a parameter.</p>
	<p>Replays the script.</p>
<Difference List>	<p>Displays the differences between the Record and Replay snapshots.</p> <ul style="list-style-type: none"> ▶ Correlated. Indicates whether an item was correlated or not. Correlated items have a check. ▶ Text in recording snapshot. The correlation string from the recording phase. ▶ Text in replay snapshot. The correlation string from the replay phase. ▶ First occurs in. The section of the script in which the correlation was first detected.
Correlation Options	<p>Allows you to set some advanced correlation options in the Correlation tab of the General Options dialog box. For user interface details, see "General Options Dialog Box" on page 66.</p>
Show differences in	<p>You can display all the differences in the script or only those for the current step or action.</p>

Output Window - Generation Log Tab

Displays a summary of the script's settings used for generating the code such as the recorder version and the recording option values.

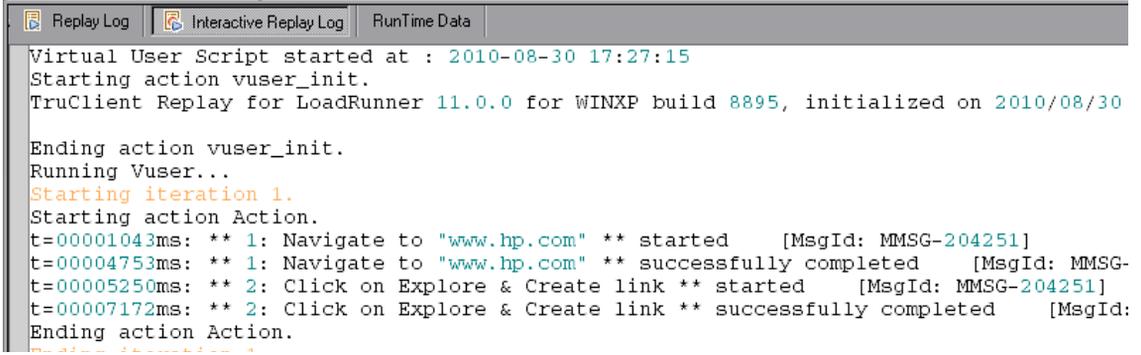


```
***** Start Log Message *****
Web Recorder version : 8.0.0.1199
Recording Mode: HTML-based script
Correlation is enabled system wide
Not recording headers
Not excluding content types
Do not record this content types as a resource:
    text/html
    text/xml
Reset context between actions
Record non-HTML elements in current HTML function
Protocol Type: Multi-QTWeb
$$$$$$ End Log Message $$$$$$
```

To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

Output Window - Interactive Replay Log Tab

For TruClient scripts only, displays messages that describe the actions of the Vuser as it runs as well as any errors that occurred.



```

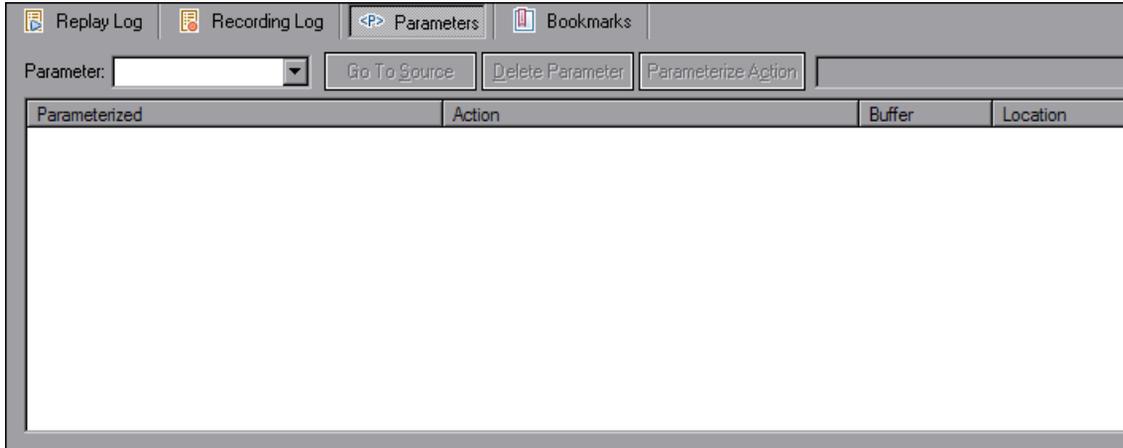
Replay Log | Interactive Replay Log | RunTime Data
Virtual User Script started at : 2010-08-30 17:27:15
Starting action vuser_init.
TruClient Replay for LoadRunner 11.0.0 for WINXP build 8895, initialized on 2010/08/30
Ending action vuser_init.
Running Vuser...
Starting iteration 1.
Starting action Action.
t=00001043ms: ** 1: Navigate to "www.hp.com" ** started [MsgId: MMSG-204251]
t=00004753ms: ** 1: Navigate to "www.hp.com" ** successfully completed [MsgId: MMSG-204251]
t=00005250ms: ** 2: Click on Explore & Create link ** started [MsgId: MMSG-204251]
t=00007172ms: ** 2: Click on Explore & Create link ** successfully completed [MsgId: MMSG-204251]
Ending action Action.

```

To access	View > Output Window
Relevant tasks	"How to Debug Ajax TruClient Scripts" on page 517

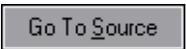
Output Window - Parameters Tab (Winsock only)

This tab allows WinSocket scripts to create and manage correlations.



To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

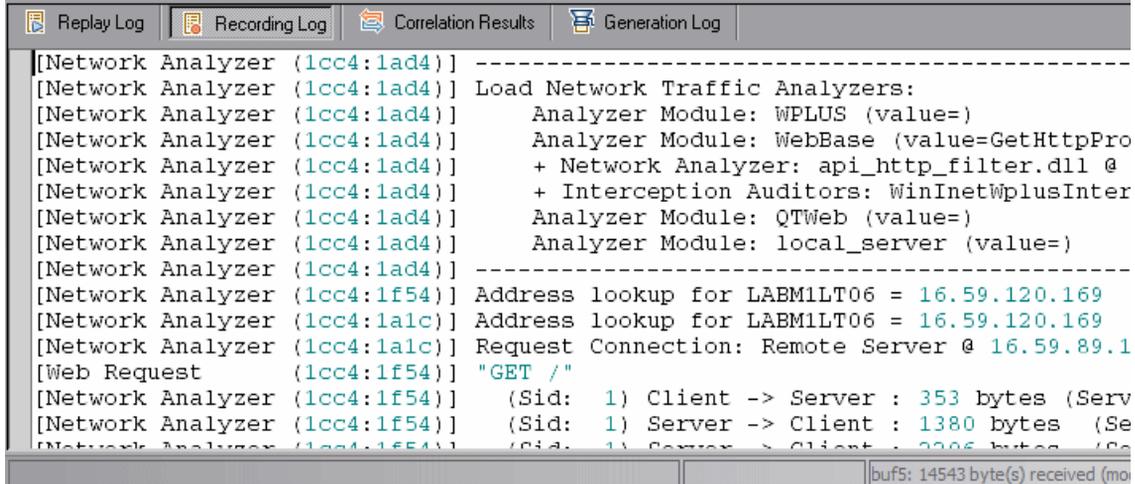
User interface elements are described below:

UI Elements (A-Z)	Description
Parameter	Select the desired parameter from the drop down menu.
<Parameter Instance List>	<p>Lists the instances of the selected parameter and provides the following details for each instance:</p> <ul style="list-style-type: none"> ▶ Parameterized. Indicates whether this instance has been parameterized or not. To change the value, right-click the instance and select Undo Replacement or Replace with Parameter. ▶ Action. The section of the script in which the instance is located. ▶ Buffer. The buffer that this instance is located in. ▶ Location. The location of the instance within the buffer. ▶ Description. A description of the status of the instance. This changes when the value of the parameterize column is modified.
	Deletes the selected parameter from the script. When you delete a parameter, VuGen replaces the data with its original value and removes the parameterization function from the script.
	Jumps to the selected instance of the parameter.
	<p>Jumps to the source data for the parameter.</p> <p>Note: This does not work if there is not value in the Action column of the Parameter Instance List.</p>
	If you recorded your script in multiple sections (e.g. init, action, and end) and you create a parameter, instances are only found for the script section that is currently selected. To find instances of the parameter in other sections, select the section and click the Parameterize Action button.
	Parameterize all instances of the selected parameter.

UI Elements (A-Z)	Description
	Undo a replacement of the selected parameter.
	Undo all replacements of the selected parameter.

Output Window - Recording Log Tab

Displays messages that were issued during recording. Depending on which protocol you are using, you may be able to set the level of detail for this log in the Recording Options.



```

[Network Analyzer (1cc4:1ad4)] -----
[Network Analyzer (1cc4:1ad4)] Load Network Traffic Analyzers:
[Network Analyzer (1cc4:1ad4)]     Analyzer Module: WPLUS (value=)
[Network Analyzer (1cc4:1ad4)]     Analyzer Module: WebBase (value=GetHttpPro
[Network Analyzer (1cc4:1ad4)]     + Network Analyzer: api_http_filter.dll @
[Network Analyzer (1cc4:1ad4)]     + Interception Auditors: WinInetWplusInter
[Network Analyzer (1cc4:1ad4)]     Analyzer Module: QTWeb (value=)
[Network Analyzer (1cc4:1ad4)]     Analyzer Module: local_server (value=)
[Network Analyzer (1cc4:1ad4)] -----
[Network Analyzer (1cc4:1f54)] Address lookup for LABM1LT06 = 16.59.120.169
[Network Analyzer (1cc4:1a1c)] Address lookup for LABM1LT06 = 16.59.120.169
[Network Analyzer (1cc4:1a1c)] Request Connection: Remote Server @ 16.59.89.1
[Web Request (1cc4:1f54)] "GET /"
[Network Analyzer (1cc4:1f54)] (Sid: 1) Client -> Server : 353 bytes (Serv
[Network Analyzer (1cc4:1f54)] (Sid: 1) Server -> Client : 1380 bytes (Se
[Network Analyzer (1cc4:1f54)] (Sid: 1) Server -> Client : 3206 bytes (Se
buf5: 14543 byte(s) received (mo
    
```

To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

📌 Output Window - Replay Log Tab

Displays messages that describe the actions of the Vuser as it runs as well as any errors that occurred.

```

Virtual User Script started at : 2010-06-03 09:58:49
Starting action vuser_init.
Web Turbo Replay of LoadRunner 11.0.0 for WINXP; build 6252 (May 25 2010 00:5
Run Mode: HTML [MsgId: MMSG-26000]
Run-Time Settings file: "C:\Program Files\HP\LoadRunner\scripts\Correlation E
Ending action vuser_init.
Running Vuser...
Starting iteration 1.
Starting action Action.
Action.c(4): Found resource "http://lazarus/Image2.gif" in HTML "http://lazar
Action.c(4): web_url("lazarus") was successful, 8916 body bytes, 544 header b
Action.c(14): web_url("All kind of CGI's.") was successful, 2637 body bytes,
Action.c(24): web_url("Link to page of useful CGIs") was successful, 3444 bod
Action.c(34): web_url("Simple Correlation ASP") was successful, 1347 body byt
Action.c(44): Found resource "http://lazarus/correlation.asp?product=link_fa
buf5: 14543 byte(s) received (mo

```

To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

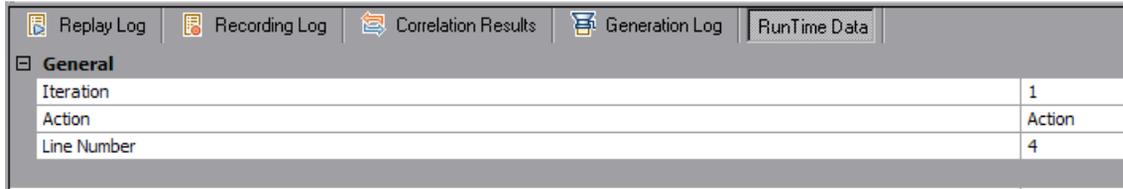
Various colors of text are used in the Replay Log to help you identify the different types of messages.

Double-clicking on a line beginning with the Action name will cause the cursor to jump to the relevant step within the script.

You can configure the amount of detail that is sent to the replay log in the **Run Time Settings > General > Log** node. For more information see "General Log Node" on page 438.

Output Window - Run Time Data Tab

The RunTime Data tab is only accessible during replay.

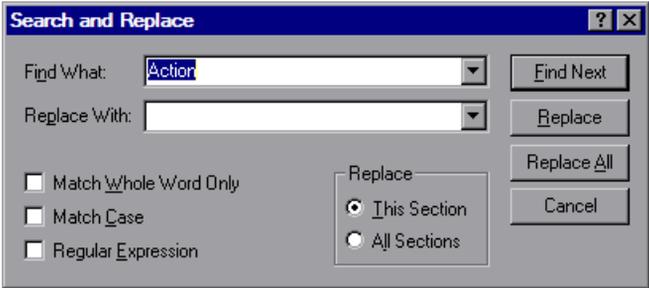


To access	View > Output Window
Relevant tasks	"How to Replay a Vuser Script" on page 129

- ▶ **General.** Displays the current iteration number, the Action name of the currently replayed step, and the line number within the script (Script view).
- ▶ **Parameters.** Displays 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 about parameters, see "Parameters" on page 257.

Search and Replace Dialog Box

This dialog box enables you to find and replace strings in the VuGen code.



To access	Edit > Replace
-----------	----------------

User interface elements are described below:

UI Elements (A-Z)	Description
Find What	Specify the words to search for.
Replace With	Type the text that will replace the text specified in the Find What box.
Match Whole Word Only	Searches for occurrences that are whole words, and not part of a larger word.
Match Case	Distinguishes between uppercase and lowercase characters during the search.
Regular Expression	Indicates that the search string is regular expression.
Replace	<ul style="list-style-type: none"> ▶ This Section. Replace occurrences in the selected section. ▶ All Sections. Replace occurrences in the entire script.
Find Next	Finds the next occurrence of the text in the Find What box.

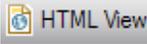
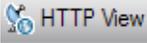
UI Elements (A-Z)	Description
	Finds the text in the Find What box and replaces it with the text in the Replace With box for the highlighted occurrence.
	Finds the text in the Find What box and replaces it with the text in the Replace With box according to the selection of This Section or All Sections described above.

Snapshot Pane

This section displays data about the selected script steps.

To access	Tree View, pane on right.
Important information	THIS PANE APPEARS DIFFERENT DEPENDING ON THE TYPE OF STEP THAT IS SELECTED.

Web (HTTP/HTML) Protocol

UI Elements (A-Z)	Description
	Displays step data in HTML format.
	Displays step data in HTTP format. This allows the user to view in depth information about the step including request data, response data, cookies, and headers. For more information, see "Web Snapshots" on page 894
	(HTTP View only) Displays the HTTP flow data in a list format.
	(HTTP View only) Displays the HTTP flow data in a tree structure.
	Displays the snapshot data from the recording phase.

UI Elements (A-Z)	Description
	Displays snapshot data for both the recording and replay phases.
	Displays the snapshot data from the replay phase.

Flex and AMF Protocols

The snapshot tab displays the step data in a number of different formats. You can view step data from the recording or replay phases. You can view request or response data. You can view the data in tree view or xml format.

 **Troubleshooting and Limitations**

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

Troubleshooting Snapshots

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

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

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

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

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

- ▶ The script was recorded with VuGen version 6.02 or earlier.

- ▶ The imported actions do not contain snapshots.
- ▶ The Vuser files are stored in a read-only directory, and VuGen could not save the replay snapshots.
- ▶ The step represents navigation to a resource.

2

Protocol Advisor

This chapter includes:

Concepts

- ▶ Protocol Advisor Overview on page 94

Tasks

- ▶ How to use the Protocol Advisor on page 95

Reference

- ▶ Protocol Advisor User Interface on page 98

Troubleshooting and Limitations on page 99

Concepts

Protocol Advisor Overview

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

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

Tasks

How to use the Protocol Advisor

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

This task includes the following steps:

- "Start the Protocol Advisor" on page 95
- "Perform typical business processes" on page 95
- "Save the results" on page 95
- "Select a protocol and create a new Vuser script" on page 96
- "Enhance, debug, and verify replay" on page 96
- "If unsuccessful replay, select a different protocol and repeat" on page 97

1 Start the Protocol Advisor

From the start page, select **File > Protocol Advisor > Analyze Application** and fill in the dialog box.

2 Perform typical business processes

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

3 Save the results

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

4 Select a protocol and create a new Vuser script

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

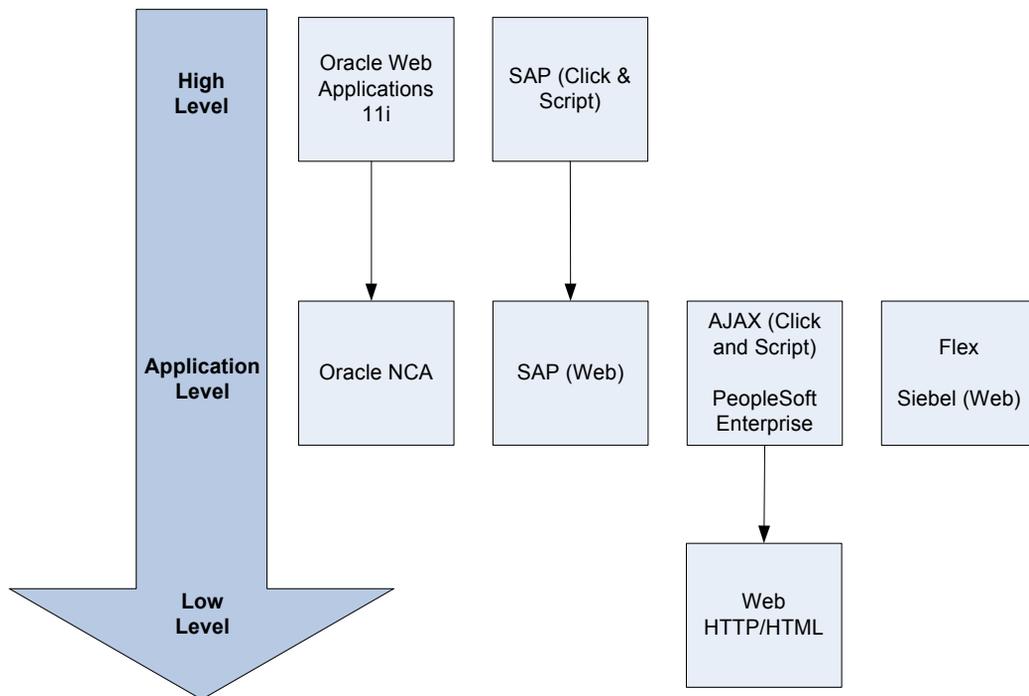
- Multi/Combination protocol
- The highest level application protocol (see the diagram in step 6 on page 97)
- The first protocol on the list

5 Enhance, debug, and verify replay

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

6 If unsuccessful replay, select a different protocol and repeat

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



Reference

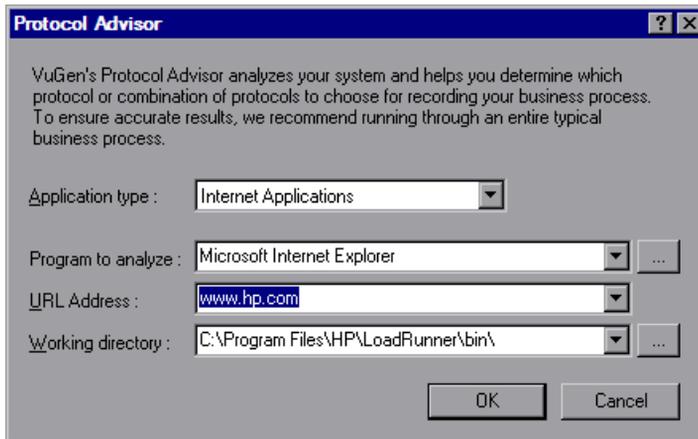
Protocol Advisor User Interface

This section includes:

- ▶ Protocol Advisor Dialog Box on page 98

Protocol Advisor Dialog Box

This dialog box enables you to open your application using the Protocol Advisor in order to determine the appropriate VuGen protocol to use for recording.



To access	File > Protocol Advisor > Analyze Application
Important information	The options in the dialog box change according the selection you make in the Application type field.
Relevant tasks	"How to use the Protocol Advisor" on page 95

User interface elements are described below:

UI Elements (A-Z)	Description
Application type	The application type: Win32 applications or internet applications.
Program arguments (Win32 Applications only)	Specify command line arguments for the executable specified above. For example, if you specify <i>plus32.exe</i> with the command line options <i>peter@neptune</i> , it connects the user <i>Peter</i> to the server <i>Neptune</i> when starting <i>plus32.exe</i> . This options is only displayed for Win32 applications.
Program to analyze	Select the browser, internet application, or Win32 application to analyze.
URL Address (Internet Applications only)	The starting URL address. This options is only displayed for internet applications.
Working directory	For applications that require you to specify a working directory, specify it here.

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Protocol Advisor.

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

- WinSocket
- LDAP

3

Recording

This chapter includes:

Concepts

- ▶ Providing Authentication Information on page 102
- ▶ Script Directory Files on page 104
- ▶ Vuser Script Templates on page 104
- ▶ Script Sections on page 105

Tasks

- ▶ How to Create or Open a Vuser Script on page 107
- ▶ How to Work with .zip Files on page 108
- ▶ How to Import Code from a Script on page 110
- ▶ How to Record a Vuser Script on page 110
- ▶ How to Regenerate a Vuser Script on page 111
- ▶ How to Create and Open Vuser Script Templates on page 113

Reference

- ▶ Files Generated During Recording on page 114
- ▶ Recording User Interface on page 116

Concepts

Providing Authentication Information

The following section only applies to multi-protocol scripts.

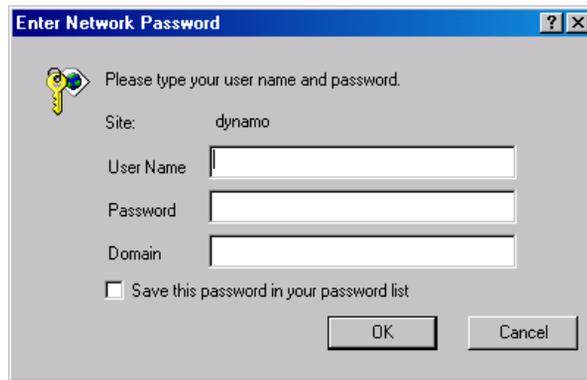
When recording a Web session that uses NTLM authentication, your server may require you to enter details such as a user name and password.

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

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



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



Generating a `web_set_user` function

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

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

```
web_set_user("domain1\dashwood",
  Ir_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 142.

Script Directory Files

While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script. To open the script folder, switch to Script view (**View > Script View**) and select **Open Script Directory** from the right-click menu.

Vuser Script Templates

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

The template supports the following files and data:

- Run-Time Settings
- parameters
- extra files
- actions
- snapshots

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

Notes and Limitations

- ▶ Once you have configured a script for a specific protocol and then save the script as a template, further scripts based on that template will only work with that same protocol. For example, if you configured your script for the Web (Click 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.

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...
<code>vuser_init</code>	a login to a server	the Vuser is initialized (loaded)
<i>Actions</i>	client activity	the Vuser is in Running status
<code>vuser_end</code>	a logoff procedure	the Vuser finishes or is stopped

When you run multiple iterations of a Vuser script, only the *Actions* sections of the script are repeated—the `vuser_init` and `vuser_end` sections are not repeated. For more information on the iteration settings, see "General Run Logic Node" on page 451.

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

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

For more information, see "Java Protocol - Manually Programming Scripts" on page 701.

```
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.

Tasks

How to Create or Open a Vuser Script

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

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

This task includes the following steps:

- "Use the protocol advisor to select a script type" on page 108
- "Create a new single protocol script" on page 108
- "Create a new multiple protocol script" on page 108
- "Create or open a script from a template" on page 108
- "Open a regular script" on page 108
- "Open or work with a .zip script" on page 108
- "Open a script stored in Application Lifecycle Management" on page 108

Use the protocol advisor to select a script type

You can use the protocol advisor to produce a list of relevant types of Vuser scripts based on your application. For details, see "Protocol Advisor" on page 93.

Create a new single protocol script

Select **File > New > New Single Protocol Script** and select the type of protocol. For user interface details, see "New Virtual User Dialog Box" on page 117.

Create a new multiple protocol script

Select **File > New > New Multiple Protocol Script** and move the desired protocols from the **Available Protocols** list to the **Selected Protocols** list. For user interface details, see "New Virtual User Dialog Box" on page 117.

Create or open a script from a template

For task details, see "How to Create and Open Vuser Script Templates" on page 113.

Open a regular script

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

Open or work with a .zip script

You can unzip or work with a script in .zip format. For task details, see "How to Work with .zip Files" on page 108.

Open a script stored in Application Lifecycle Management

You can store scripts on HP ALM and modify them in VuGen. For more information, see "Working with Application Lifecycle Management" on page 245.

How to Work with .zip Files

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

This task includes the following steps:

- "Import from a .zip file" on page 109
- "Work from a .zip file" on page 109
- "Save scripts in .zip format" on page 109
- "Compress and email a script" on page 110

Import from a .zip file

To open a script stored in a .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.

Work 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.

Save scripts in .zip format

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 or only runtime and specify the compression ratio. The greater the compression ratio, the longer VuGen takes to create the archive (**Maximum** compression option is the slowest).

Compress and email a script

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.

How to Import Code from a Script

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

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.

How to Record a Vuser Script

This task describes how to record a Vuser script.

This task includes the following steps:

- "Configure the recording options - optional" on page 111
- "Initialize the recording session" on page 111
- "Perform a business process on your application" on page 111

1 Configure the recording options - optional

The recording options contain many different options that affect your script during the recording and generation stages of creating a script. For concept and user interface details, see "Recording Options" on page 307.

2 Initialize the recording session

When creating a new script, this occurs automatically. To start recording manually, click the **Start Record** button in the toolbar and complete the Start Recording dialog box. For user interface details, see "Start Recording Dialog Box" on page 119.

3 Perform a business process on your application

VuGen will automatically open your application as well as a floating toolbar and begin recording your actions. Perform the desired business processes that you wish to record. The toolbar allows you to insert transactions, rendezvous points, comments, as well as determine which section of the script to record in while recording your application. For user interface details, see "Main VuGen User Interface" on page 72.



Click the **stop** button on the toolbar when you are finished.

How to Regenerate a Vuser Script

If you need to revert back to your originally recorded script, you can regenerate it. This feature is ideal for debugging, or fixing a corrupted script. When you regenerate a script, 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.

This task describes how to regenerate a Vuser script.

This task includes the following steps:

- "Initialize the regeneration" on page 112
- "Modify regenerate options - optional" on page 112

1 Initialize the regeneration

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

2 Modify regenerate options - optional

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

In a multiple protocol script, you can modify which protocols you want to record when the script is regenerated from the **General > Protocols** node. For user interface details, see "General Protocol Node" on page 348.

To change the Script options, select the **General > Script** node and select or clear the appropriate check box. For user interface details, see "General Script Node" on page 353.

How to Create and Open Vuser Script Templates

This task describes how to create and open a Vuser script template.

Create a Vuser Script Template

Open a script in VuGen, Select **File > User-Defined Template > Save As Template** and enter a name for the template.

Open a Vuser Script Template

Select **File > User-Defined template > Create Script From Template** and select the template file.

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.

Reference

Files Generated During Recording

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

File Name	Details
vuser.usr	Contains information about the virtual user: type, AUT, action files, and so forth.
vuser.bak	A copy of Vuser.usr before the last save operation.
default.cfg	Contains a listing of all run-time settings as defined in the VuGen application (think time, iterations, log, web).
vuser.asc	The original recorded API calls.
vuser.grd	Contains the column headers for grids in database scripts.
default.usp	Contains the script's run logic, including how the actions sections run.
init.c	Exact copy of the Vuser_init function as seen in the VuGen main window.
run.c	Exact copy of the Action function as seen in the VuGen main window.
end.c	Exact copy of the Vuser_end function as seen in the VuGen main window.

File Name	Details
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]
XIBridgeTimeout=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
```

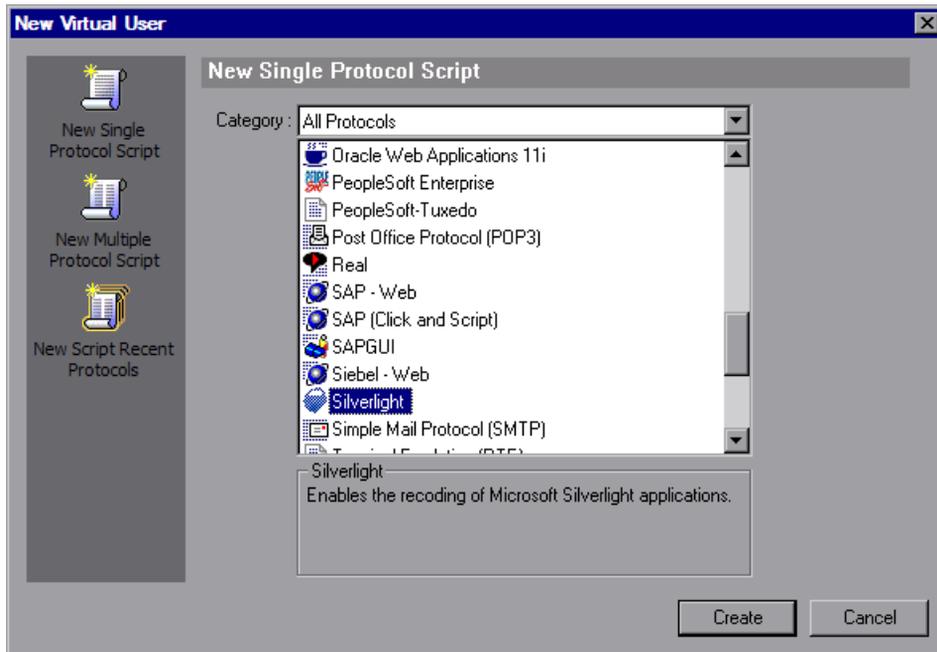
Recording User Interface

This section includes:

- ▶ New Virtual User Dialog Box on page 117
- ▶ Start Recording Dialog Box on page 119

New Virtual User Dialog Box

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



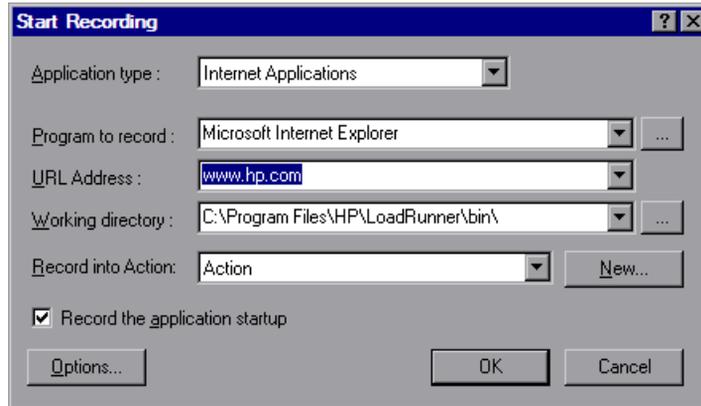
To access	File > New
Relevant tasks	<ul style="list-style-type: none"> ➤ "How to Create or Open a Vuser Script" on page 107 ➤ "How to Record a Vuser Script" on page 110

User interface elements are described below:

UI Elements (A-Z)	Description
New Single Protocol Script	Creates a new single protocol script. <ul style="list-style-type: none"> ▶ Category drop-down list. A list of protocol categories. To display all protocols select All Protocols. ▶ Protocol list. List of protocols for the given category.
New Multiple Protocol Script	Creates a new multiple protocol script. Use the arrows to move protocols from the Available Protocols to the Selected Protocols list. Selecting Create creates a new multiple protocol script based on the protocols in the Selected Protocols list.
New Script Recent Protocols	Creates a new script using a recently used protocol.

Start Recording Dialog Box

This dialog box enables specify the basic details needed to begin recording a script.



To access	Opens automatically when you begin recording
Important information	This dialog box is dynamic and changes according to the options you select as well as the protocol you are using.
Relevant tasks	"How to Record a Vuser Script" on page 110

User interface elements are described below:

All Protocols (except Java)

UI Elements (A-Z)	Description
	Opens the Recording Options dialog box. For user interface details, see "Recording Options" on page 307.
Application type	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.

UI Elements (A-Z)	Description
Program arguments (Win32 Applications only)	Specify command line arguments for the executable specified above. For example, if you specify <i>plus32.exe</i> with the command line options <i>peter@neptune</i> , it connects the user <i>Peter</i> to the server <i>Neptune</i> when starting <i>plus32.exe</i> . This options is only displayed for Win32 applications.
Program to record	Select the browser, internet application, or Win32 application to record.
Record into action	The section into which you want to record.
Record the application startup	In the following instances, it may not be advisable to record the startup: <ul style="list-style-type: none"> ▶ 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.
URL Address (Internet Applications only)	The starting URL address. This options is only displayed for internet applications.
Working directory	For applications that require you to specify a working directory, specify it here.

Java Protocols

UI Elements (A-Z)	Description
	Opens the Recording Options dialog box. For user interface details, see "Recording Options" on page 307.
Working Directory	A working directory is necessary only if your application must know the location of the working directory (for example, reading property files or writing log files).
Record into action	The section into which you want to record.
URL	The URL to start recording.

UI Elements (A-Z)	Description
Application Parameters	Any additional parameters that your application requires.
App Main Class	This option is only present for Java Application type applications.
IExplore Path	This option is only present for IExplore type applications.
Executable\Batch	This option is only present for Executable\Batch type applications.
Netscape Path	This option is only present for Netscape type applications.
Application type	<ul style="list-style-type: none"> ▶ Java Applet to record a Java applet through Sun's applet viewer. ▶ 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 <code>_JAVA_OPTIONS</code> as <code>--Xrunjdkhook</code> using <code>jdk1.2.x</code> and higher. (For JDK 1.1.x, define the environment variable <code>_classload_hook=JDKhook</code>. For JDK 1.6 set <code>_JAVA_OPTIONS</code> as <code>-agentlib:jdkhook</code>.)

4

Replaying and Debugging Vuser Scripts

This chapter includes:

Concepts

- ▶ Replay Overview on page 124
- ▶ Debugging Features on page 125
- ▶ Additional Debugging Information on page 126
- ▶ Debugging Features for Web Vusers on page 127

Tasks

- ▶ How to Replay a Vuser Script on page 129
- ▶ How to Run a Vuser Script from a Command Prompt on page 130
- ▶ How to Run a Vuser Script from a UNIX command line on page 131
- ▶ How to Debug Scripts with Breakpoints on page 133
- ▶ How to Use Bookmarks on page 135

Reference

- ▶ Files Generated During Replay on page 137
- ▶ Replay User Interface on page 139

Concepts

Replay Overview

After creating a script, you replay it in standalone mode, directly from the VuGen interface. This tests the basic functionality as well as helps you to uncover any errors or issues that need to be addressed such as dynamic values which need parameterization.

When the replay is successful, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the relevant user guide.

Debugging Features

VuGen contains the following features to help debug Vuser scripts. They are not available for VBScript and VB Application type Vusers. You access these features from the Debug toolbar. To view this toolbar, right-click the toolbar area and select **Debug**.

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, select **Vuser > Run Step by Step** and continue 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. For task details, see "How to Debug Scripts with Breakpoints" on page 133.

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. For task details, see "How to Use Bookmarks" on page 135.

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.

Additional Debugging Information

General Debugging Tip

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

Using C Functions for Tracing

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

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

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

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

Additional C Language Keywords

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

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 Features for Web Vusers

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 more user interface information, see "Display Tab" on page 100.

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.

Tasks

How to Replay a Vuser Script

This task describes how to replay a Vuser script.

This task includes the following steps:

- "Configure the replay options" on page 129
- "Configure the run-time settings" on page 130
- "Replay the script" on page 130
- "View the logs for detailed information" on page 130

1 Configure the replay options

Specify the desired options in the replay and display tabs of the General Options dialog box. For user interface details, see "General Options Dialog Box" on page 95.

2 Configure the run-time settings

The Run-time settings dialog box contains settings that affect the way your script behaves when running or replaying. For user interface details, see "Run-Time Settings" on page 417.

3 Replay the script

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

4 View the logs for detailed information

You can view detailed information about how your script behaved during the recording and replay stages in the output window. For user interface details, see "Output Window" on page 79.

How to Run a Vuser Script from a Command Prompt

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

To run a script from a command line or the Run dialog box:

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

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

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

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

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

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

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

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

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

How to Run 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. This task describes how to perform this check and run a Vuser script from a UNIX command.

1 Verify that the script replays in VuGen

Replay the script in VuGen to verify that the script works in windows before attempting to run it in UNIX. This is recommended because it is easier to edit and debug the script in VuGen. For task details, see "How to Replay a Vuser Script" on page 129.

2 Copy the script files to the UNIX server

Transfer the script files to the UNIX server.

3 Check the Vuser setup on the UNIX machine by using **verify_generator**.

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

```
verify_generator
```

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

For detailed information about the verify checks type:

```
verify_generator [-v]
```

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

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

4 Run the script

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

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

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

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

Command	Description
-driver <i>driver_path</i>	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 <i>ci_file_name</i>	Execute a specific .ci file.
-out <i>output_path</i>	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
```

How to Debug Scripts with Breakpoints

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

- "Add a breakpoint" on page 134
- "Enable/Disable a breakpoint" on page 134
- "Manage breakpoints" on page 134
- "Run a script with breakpoints" on page 134

Add a breakpoint



Select **Insert > Toggle Breakpoint**, or click the **Breakpoint** button in the Debug toolbar. The Breakpoint symbol (●) appears in the left margin of the script.

Enable/Disable a breakpoint



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.

Manage breakpoints

The Breakpoint dialog box allows you to add, remove, enable, disable, and conditionalize all of your breakpoints in one user interface. For user interface details, see "Breakpoints Dialog Box" on page 139.

Run a script with breakpoints

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

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

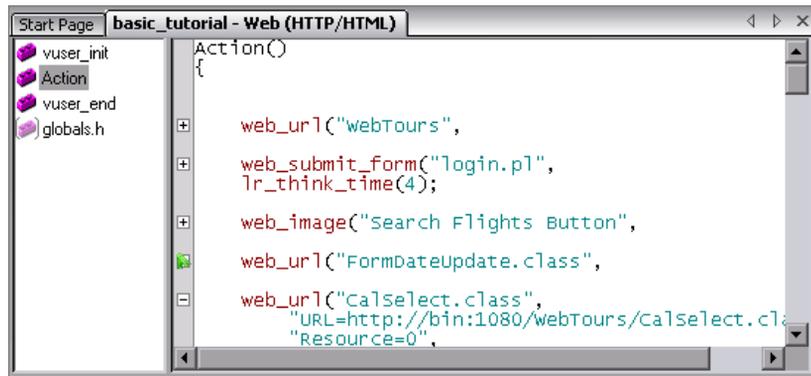
How to Use Bookmarks

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. The following steps describe how to work with bookmarks.

- "Create a bookmark" on page 135
- "Remove a bookmark" on page 135
- "Navigate between bookmarks" on page 135

Create a bookmark

Place the cursor at the desired location and press Ctrl + F2. VuGen places an icon in the left margin of the script.



Remove a bookmark

To remove a bookmark, click on the desired bookmark and press Ctrl + F2. VuGen removes the bookmark icon from the left margin.

Navigate between bookmarks

- To move to the next bookmark, click **F2**.
- To return to the previous bookmark, click **Shift + F2**.

You can also create and navigate between bookmarks through the **Edit > Bookmarks** menu item.

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.

Reference

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  
-lc:\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"
```

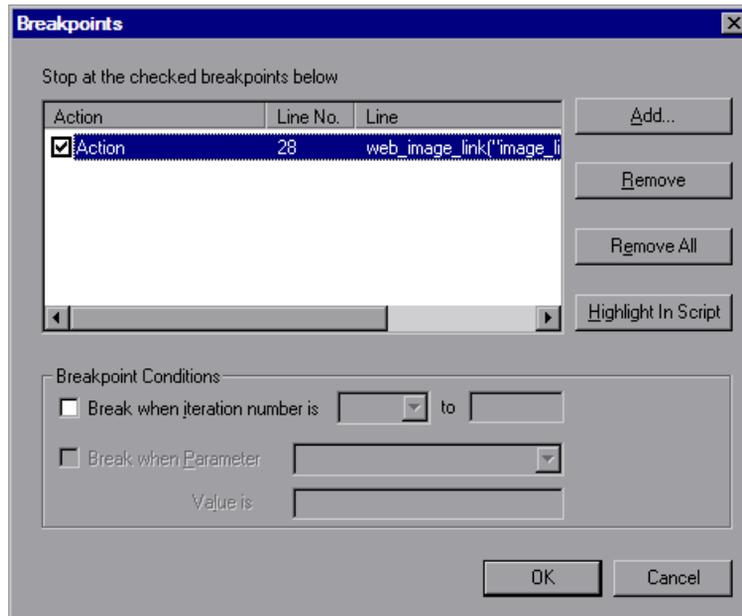
Replay User Interface

This section includes:

- Breakpoints Dialog Box on page 139

Breakpoints Dialog Box

This dialog box enables you to manage breakpoints.



To access	Edit > Breakpoints
Relevant tasks	"How to Debug Scripts with Breakpoints" on page 133

User interface elements are described below:

UI Elements (A-Z)	Description
	Adds a new breakpoint at the specified section and line number.
	Removes the selected breakpoint.
	Removes all breakpoints.
	Displays the script with the breakpoint highlighted. You can only highlight one breakpoint at a time.
<Breakpoints Grid>	A list of current breakpoints and their locations in the script. To enable a breakpoint, select the checkbox next to that breakpoint. To disable a breakpoint, clear the checkbox.
Breakpoint Conditions	<ul style="list-style-type: none"> ▶ Break when iteration number is. Pauses the script after a specific number of iterations. ▶ Break when parameter value is. Pause the script when parameter X has a specific value.

5

Preparing Scripts for Load Testing

This chapter includes:

Concepts

- ▶ Password Encoding on page 142
- ▶ Encrypting Text on page 142
- ▶ Transaction Overview on page 143
- ▶ Rendezvous Points on page 144
- ▶ Think Time on page 145

Tasks

- ▶ How to Encrypt/Decrypt Text on page 146
- ▶ How to Encode a Password on page 147
- ▶ How to Create a Controller Scenario from VuGen on page 147
- ▶ How to Insert Transactions on page 148
- ▶ How to Display Transactions on page 150
- ▶ How to Prepare a Script for Load Testing on page 151
- ▶ How to Insert Steps into a Script on page 155

Reference

- ▶ Useful VuGen Functions on page 157
- ▶ Preparing Scripts for Load Test User Interface on page 159

Concepts

Password Encoding

You can encode passwords in order to use the resulting strings as arguments in your script or parameter values. For example, your Web site may include a form in which the user must supply a password. You may want to test how your site responds to different passwords, but you also want to protect the integrity of the passwords. The **Password Encoder** enables you to encode your passwords and place secure values into the table.

To encode a password, select **Start > Programs > LoadRunner > Tools > Password Encoder**.

For task details, see "How to Encode a Password" on page 147.

For user interface details, see "Password Encoder" on page 162.

Encrypting Text

You can encrypt text within your script to protect your passwords and other confidential text strings. You can perform encryption both automatically, from the user interface, and manually, through programming. You can restore the string at any time, to determine its original value. When you encrypt a string, it appears in the script as a coded string. Note that VuGen uses 32-bit encryption.

In order for the script to use the encrypted string, it must be decrypted with **lr_decrypt**.

```
lr_start_transaction(lr_decrypt("3c29f4486a595750"));
```

For task details, see "How to Encrypt/Decrypt Text" on page 146.

Transaction Overview

You define *transactions* to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified Vuser requests. These requests can be simple tasks such as waiting for a response for a single query, or complex tasks, such as submitting several queries and generating a report.

To measure a transaction, you insert Vuser functions to mark the beginning and the end of a task. Within a script, you can mark an unlimited number of transactions, each transaction with a different name.

For LoadRunner, the Controller measures the time that it takes to perform each transaction. After the test run, you analyze the server's performance per transaction using the Analysis' graphs and reports.

Before creating a script, you should determine which business processes you want to measure. You then mark each business process or sub-process as a transaction.

Transaction names cannot contain a "_" or "@" symbol. This will cause errors to occur when attempting to open the Analysis Cross Results graphs.

You can create transactions either during or after recording. For task details, see "How to Insert Transactions" on page 148.

Rendezvous Points

When performing load testing, you need to emulate heavy user load on your system. To accomplish this, you synchronize Vusers to perform a task at exactly the same moment. You configure multiple Vusers to act simultaneously by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it waits until all Vusers participating in the rendezvous arrive. When the designated number of Vusers arrive, the Vusers are released.

You designate the meeting place by inserting a rendezvous point into your Vuser script. When a Vuser executes a script and encounters the rendezvous point, script execution is paused and the Vuser waits for permission from the Controller to continue. After the Vuser is released from the rendezvous, it performs the next task in the script.

For task details, see "Insert Rendezvous Points" on page 152.

Note: Rendezvous points are only effective in *Action* sections—not *init* or *end*.

 **Think Time**

The time that a user waits between performing successive actions is known as the *think time*. Vusers use the **lr_think_time** function to emulate user think time. When you record a Vuser script, VuGen records the actual think times and inserts appropriate **lr_think_time** statements into the Vuser script. You can edit the recorded **lr_think_time** statements, and manually add more **lr_think_time** statements to a Vuser script.

Note: You can insert a think time step by selecting **Insert > Step > Think Time**. For task details, see "Insert Think Time Steps" on page 156. When you record a Java Vuser script, **lr_think_time** statements are not generated in the Vuser script.

You can use the run-time settings to influence how the **lr_think_time** statements operate when you execute a Vuser script. For user interface details, see "General Think Time Node" on page 452.

Tasks

How to Encrypt/Decrypt Text

This task describes how to encrypt and decrypt strings in your code. For background information, see "Encrypting Text" on page 142.

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**).

How to Encode a Password

This task describes how to encode a password. You can encode passwords in order to use the resulting strings as arguments in your script or parameter values. For example, your Web site may include a form in which the user must supply a password. You may want to test how your site responds to different passwords, but you also want to protect the integrity of the passwords.

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.

How to Create a Controller Scenario from VuGen

Note: The following section only applies to LoadRunner. For information on integrating scripts into Business Process Manager configurations, see the *HP Business Service Management* documentation.

Normally, you create a scenario from the LoadRunner Controller. You can also create a basic scenario from VuGen using the current script.

To create this type of scenario, select **Tools > Create Controller Scenario** and complete the dialog box. For user interface details, see "Create Scenario Dialog Box" on page 160.

For more information, see the *HP LoadRunner Controller User Guide*.

How to Insert Transactions

Notes:

- ▶ You can create *nested* transactions—transactions within transactions. If you nest transactions, close the inner transactions before closing the outer ones—otherwise the transactions won't be analyzed properly. However, transactions must be contained within a single **action** section.
- ▶ Transaction names must be unique and begin with a letter or number and may contain letters or numbers. Do not use the following characters: `. , : # / \ " <`.

The following steps describe different methods to insert transactions. For background information, see "Transaction Overview" on page 143.

- ▶ "Insert transactions during recording" on page 149
- ▶ "Insert transactions in the script or tree view" on page 148
- ▶ "Insert transactions using the Transaction Editor" on page 149

Insert transactions in the script or tree view

- ▶ To mark the start of a transaction after recording, select **Insert > Start Transaction** and enter a transaction name.
- ▶ To mark the end of a transaction, select **Insert > End Transaction**. VuGen inserts an `lr_end_transaction` statement into the Vuser script.

Insert transactions using the Transaction Editor

Notes:

- This option is only available for scripts that have thumbnails.
 - 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, or transactions with errors. Therefore, you might see a caption similar to the following caption above the transaction list: **Transactions (2 of 4)**. To display all transactions, see "How to Display Transactions" on page 150.
-
- To mark a multiple step transaction, select **Transactions** from the task pane and select **New Transaction**. Drag the cursor to a location in front of the thumbnail you want to be the start of the transaction and click. Place the cursor in front of the thumbnail you want to be the end of the transaction and click. Enter a name for the transaction by clicking on the title.
 - To mark a single step 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.
 - To change the starting point of a transaction, drag the transaction opening bracket to a new location. To change the ending point of a transaction, drag the transaction closing bracket to a new location.
 - Use the right-click menu to add, rename, and delete transactions.
- For user interface details, see "Transaction List" on page 163.

Insert transactions during recording



- To mark the start of a transaction, click the **Start Transaction** button on the Recording toolbar while recording a script and enter a transaction name. When you click **OK**, VuGen inserts an `lr_start_transaction` statement into the Vuser script.



- To mark the end of a transaction, click the **End Transaction** button on the Recording toolbar and select the transaction to close. When you click **OK**, VuGen inserts an **lr_end_transaction** statement into the Vuser script.

How to Display Transactions

The following steps describe how to display different types of transactions when viewing them in the task pane. For background information, see "Transaction Overview" on page 143.

- "Display hidden transactions" on page 151
- "Display transactions with errors" on page 151
- "Display transactions for non-primary steps" on page 151

Display hidden transactions

To display the hidden transactions—the non-primary and client side transactions—click the button adjacent to **Show hidden transactions** at the bottom of the transaction list. VuGen lists the hidden transactions in gray. To hide them, click the button again.

Display transactions with errors

Transactions with errors are those that do not measure any server steps, or those with illegal names. To show the transactions with errors, click the **Show transactions with errors** button. VuGen lists the transactions with errors in red. To hide them, click the button again.

Display transactions for non-primary steps

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

How to Prepare a Script for Load Testing

This task describes the additional things you can do to your script after you have debugged it to prepare it for load testing. All of the items in this task are optional.

This task includes the following steps:

- "Insert Steps" on page 152
- "Edit Steps" on page 152
- "Insert Transactions" on page 152
- "Insert Rendezvous Points" on page 152
- "Insert Comments" on page 153
- "Insert VuGen Functions" on page 153
- "Insert Log Messages" on page 154
- "Add Files to the Script" on page 154

- ▶ "Synchronize the Script (RTE Vusers only)" on page 154
- ▶ "Configure the Test Results Window Options" on page 155
- ▶ "Replay and Debug your Script" on page 155

Insert Steps

You can insert a variety of steps into your script such as think time steps, debug messages, and output messages. For task details, see "How to Insert Steps into a Script" on page 155.

Edit Steps

You can modify individual steps by right-clicking the step in tree view and selecting **properties**.

Insert Transactions

You can insert transactions into your script by using the **Insert > Start Transaction** and **Insert > End Transaction** menu items. For task details, see "How to Insert Transactions" on page 148.

Insert Rendezvous Points

You can synchronize Vusers to perform a task at exactly the same moment by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it waits until all Vusers participating in the rendezvous arrive. When the designated number of Vusers arrive, the Vusers are released.

You can insert rendezvous points in one of the following ways:

- ▶ To insert a rendezvous point while recording, click the **Rendezvous** button on the Recording toolbar and enter a name in the dialog box (not case sensitive).
- ▶ To insert a rendezvous point after recording, select **Insert > Rendezvous** and enter a name in the dialog box (not case sensitive).



When a rendezvous point is inserted, VuGen inserts a **lr_rendezvous** function into the Vuser script. For example, the following function defines a rendezvous point named rendezvous1:

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

For concept details, see "Rendezvous Points" on page 144.

Insert Comments

VuGen allows you to insert comments between Vuser activities. You can insert a comment to describe an activity or to provide information about a specific operation. For example, if you are recording database actions, you could insert a comment to mark the first query, such as "This is the first query."

You can insert a comment in one of the following ways:



- To insert a comment while recording, click the **Insert Comment** button on the Recording toolbar and enter the desired comment in the Insert Comment dialog box.
- To insert a comment after recording, select **Insert > Comment** and enter the desired comment in the Insert Comment dialog box.

The following script segment shows how a comment appears in a Vuser script:

```
/*
 * This is a comment
 */
```

Insert VuGen Functions

You can insert VuGen functions at this point. For a list of some useful functions see "Useful VuGen Functions" on page 157.

Insert Log Messages

You can use VuGen to generate and insert **lr_log_message** functions into a Vuser script. For example, if you are recording database actions, you could insert a message to indicate the first query, "This is the first query."

To insert a log message, select **Insert > Log Message** and enter the message.

Add 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 to the script, select **File > Add Files to Script**.

Synchronize the Script (RTE Vusers only)

You can add synchronization functions to synchronize the execution of the Vuser script with the output from your application. Synchronization applies to RTE Vuser scripts only.

The following is a list of the available synchronization functions:

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

For details about synchronization in RTE Vuser scripts, see "RTE Synchronization Overview" on page 813

Configure the Test Results Window Options

To assist with debugging a Vuser script, you can view a report that summarizes the results of your script run. VuGen generates the report during the Vuser script execution and you view the report when script execution is complete.

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

To prevent VuGen from generating the results, choose **Tools > 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.

Replay and Debug your Script

For more information, see "Replaying and Debugging Vuser Scripts" on page 123.

How to Insert Steps into a Script

The following steps describe how to add different types of steps into a Vuser script.

- "Insert Think Time Steps" on page 156
- "Insert Debug Messages" on page 156
- "Insert Error and Output Messages" on page 156

Insert Think Time Steps

The time that a user waits between performing successive actions is known as the *think time*. Vusers use the `lr_think_time` function to emulate user think time. When you record a Vuser script, VuGen records the actual think times and inserts appropriate `lr_think_time` statements into the Vuser script. You can edit the recorded `lr_think_time` statements, and manually add more `lr_think_time` statements to a Vuser script.

To add a think time step, select **Insert > Add Step > Think Time** and specify the desired think time in seconds.

Insert Debug Messages

You can add a debug or error message using VuGen's user interface. For debug messages you can indicate the level of the text message—the message is only issued when your specified level matches the message class. You set the message class using `lr_set_debug_message`.

To insert a debug message, select **Insert > New Step > Debug Message** and complete the dialog box. For user interface details, see "Debug Message Dialog Box" on page 161.

Insert 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, select **Insert > New Step > Error Message** or **Output Message**, and enter the message. An `lr_error_message` or `lr_output_message` function is inserted at the current point in the script.

Reference

Useful VuGen Functions

This section contains useful VuGen functions that you may want to add to your script while debugging or preparing your script for load testing.

Obtaining Vuser Information

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

Function	Description
lr_get_attrib_string	Returns a command line parameter string.
lr_get_host_name	Returns the name of the machine running the Vuser script.
lr_get_master_host_name	Returns the name of the machine running the Controller. Not applicable when working with the HP Business Service Management.
lr_whoami	Returns the name of a Vuser executing the script. Not applicable when working with the HP Business Service Management.

In the following example, the **lr_get_host_name** function retrieves the name of the computer on which the Vuser is running.

```
my_host = lr_get_host_name( );
```

For more information about the above functions, see the *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 Service Management, you can use Message type functions to send error and notification messages to the Web site or Business Process Monitor log files. For example, you could insert a message that displays the current state of the Web-based application.

Note: Do not send messages from within a transaction as this may lengthen the transaction execution time and skew the transaction results.

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

Function	Description
lr_debug_message	Sends a debug message to the Output window or the Business Process Monitor log file.
lr_error_message	Sends an error message to the Output window, Test Results report, or the Business Process Monitor log files.
lr_get_debug_message	Retrieves the current message class.
lr_log_message	Sends an output message directly to the log file, <i>output.txt</i> , 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.

Function	Description
lr_set_debug_message	Sets a message class for output messages.
lr_vuser_status_message	Sends a message to the Vuser status area in the Controller. Not applicable when working with the HP Business Service Management.
lr_message	Sends a message to the Vuser log and Output window or the Business Process Monitor log files.

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

Using the **lr_output_message**, and **lr_error_message** functions, you can also send meaningful messages to the Test Results summary report. For information, see Chapter 6, "Viewing Test Results."

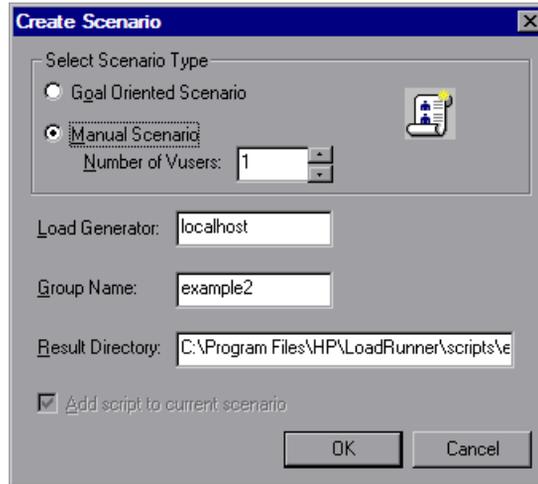
Preparing Scripts for Load Test User Interface

This section includes:

- Create Scenario Dialog Box on page 160
- Debug Message Dialog Box on page 161
- Password Encoder on page 162
- Transaction List on page 163

Create Scenario Dialog Box

This dialog box enables you to create a basic Controller scenario from within VuGen.



To access	Tools > Create Controller Scenario
Relevant tasks	"How to Create a Controller Scenario from VuGen" on page 147

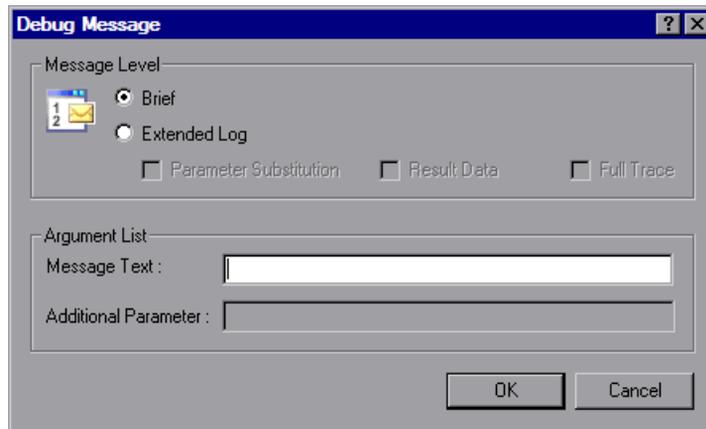
User interface elements are described below:

UI Elements (A-Z)	Description
Add script to current scenario	If a scenario is currently open in the Controller and you want to add the script to this scenario, select this check box. If you clear the check box, LoadRunner opens a new scenario with the specified number of Vusers.
Group Name	For a manual scenario, users with common traits are organized into groups. Specify a new group name for the Vusers.
Load Generator	The name of the machine that will run the scenario.

UI Elements (A-Z)	Description
Results Directory	Enter the desired location for the results.
Script Name	For a goal-oriented scenario, specify a script name.
Select Scenario Type	<ul style="list-style-type: none"> ▶ Goal Oriented Scenario. LoadRunner automatically builds a scenario based on the goals you specify. ▶ Manual Scenario. The scenario is created manually by specifying the number of Vusers to run.

Debug Message Dialog Box

This dialog box enables you to insert a debug message into your script.



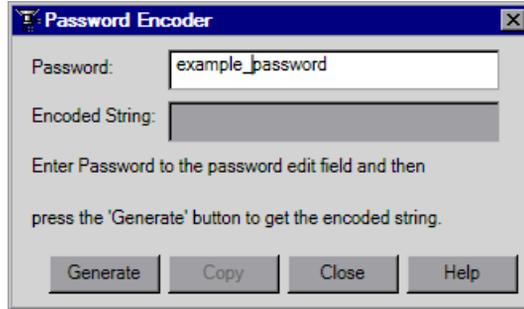
To access	Insert > New Step > Debug Message
Relevant tasks	"How to Insert Steps into a Script" on page 155

User interface elements are described below:

UI Elements (A-Z)	Description
Argument List	Enter the message in the Message Text field.
Message Level	The 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 .

Password Encoder

This dialog box enables you to generate encoded passwords.



To access	Start > Programs > LoadRunner > Tools > Password Encoder
Relevant tasks	"How to Encode a Password" on page 147
See also	"Password Encoding" on page 142

User interface elements are described below:

UI Elements (A-Z)	Description
Copy	Copy the results from the encoded string field to paste them to the Data table containing your list of parameters.
Encoded String	The encoded results are displayed here.
Generate	Click this to generate the encoded password.
Password	Enter the password you want to encode here.

Transaction List

This list enables you to manage transactions.

To access	Task Pane > Transactions
Important information	This list is only available for scripts that have thumbnails.
Relevant tasks	<ul style="list-style-type: none"> ▶ "How to Insert Transactions" on page 148 ▶ "How to Display Transactions" on page 150

User interface elements are described below:

UI Elements (A-Z)	Description
<Thumbnail List>	Displays the thumbnails with transactions labeled and marked by brackets.

UI Elements (A-Z)	Description
Action	Use this drop-down list to select the relevant section in your script.
Transaction List	<ul style="list-style-type: none"> ▶ Action. Use this drop-down list to select the relevant section in your script. ▶ Transaction List. This lists the names and the number of transactions. If the number of transactions listed is smaller than the total number of transactions (e.g. 2 of 4) there are hidden transactions. 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, or transactions with errors. Therefore, you might see a caption similar to the following caption above the transaction list: Transactions (2 of 4). To display all transactions, see "How to Display Transactions" on page 150. ▶  New Transaction . Inserts a new transaction. For task details, see "Insert transactions using the Transaction Editor" on page 149.

6

Viewing Test Results

This chapter includes:

Concepts

- ▶ Test Results Overview on page 166
- ▶ Customizing the Test Results Display on page 167
- ▶ Connecting to Application Lifecycle Management from the Test Results Window on page 168

Tasks

- ▶ How to Send Custom Information to the Report on page 169
- ▶ How to Configure the Appearance of the Test Results Window on page 170
- ▶ How to Open the Test Results of a Specific Run on page 170
- ▶ How to Find Steps in the Test Results on page 171
- ▶ How to Submit Defects to ALM on page 171

Reference

- ▶ Viewing Test Results User Interface on page 173

Concepts

Test Results Overview

To assist with debugging a Vuser script, you can view a report that summarizes the results of your script run. VuGen generates the report during the Vuser script execution and you view the report when script execution is complete.

The Test Results window displays all aspects of the test run and can include:

- ▶ a high-level results overview report (pass/fail status)
- ▶ the data used in all runs
- ▶ an expandable tree of the steps, specifying exactly where application failures occurred
- ▶ the exact locations in the script where failures occurred
- ▶ a still image of the state of your application at a particular step

- a movie clip of the state of your application at a particular step or of the entire test
- detailed explanations of each step and checkpoint pass or failure, at each stage of the test.

Customizing the Test Results Display

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

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

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

Connecting to Application Lifecycle Management from the Test Results Window

To manually submit defects to Application Lifecycle Management from the Test Results window, you must be connected to Application Lifecycle Management.

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

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

For more information on connecting to an ALM project, see "How to Work with Scripts in ALM Projects" on page 247.

Tasks

How to Send Custom Information to the Report

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

For task details, see "How to Insert Steps into a Script" on page 155.

How to Configure 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 these settings, select **View > Window Theme** and select the desired theme.

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

How to Open the Test Results of a Specific Run

This task describes how to open the test results window for a specific run.

To select a specific set of test results:

- 1** Select **File > Open** from within the Test Results window.
- 2** Select a script file to display the test results for that file and select the desired test result file. By default, result files for tests are stored in `<Script>\<ResultName>.xml`. If your script is stored in Application Lifecycle Management, see below.
- 3** Select a results set and click **Open**.

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

To select a script stored in Application Lifecycle Management:

- 1** Click the **Application Lifecycle Management Connection** button and connect to your Application Lifecycle Management project.
- 2** In the Open Test Results dialog box, enter the path of the folder that contains the results file for your QuickTest test, or click the browse button to open the Open Test from ALM Project dialog box.
- 3** Select **DB Vuser** in the **Test Type** list.
- 4** Select the script whose test results you want to view, and click **OK**.
- 5** In the Open Test Results dialog box, highlight the test result set you want to view, and click **Open**. The Test Results window displays the selected test results.

 **How to Find Steps in the Test Results**

This task describes how to search the test results for steps of a particular type.

To search the test results:

- 1** Select **Tools > Find** from within the Test Results window.
- 2** Select the type of step you wish to find. You can select multiple options.
- 3** Select **Up** or **Down** to indicate the direction of the search.
- 4** Select **Find Next** to find the next occurrence of the type of step you selected.

 **How to Submit Defects to ALM**

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

To manually submit a defect to Application Lifecycle Management:

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



2 Select **Tools > Application Lifecycle Management Connection** or click the **Application Lifecycle Management Connection** button to connect to a Application Lifecycle Management project. For more information on connecting to a project, see "Working with Application Lifecycle Management" on page 245.



3 Select **Tools > Add Defect** or click the **Add Defect** button to open the Add Defect dialog box in the specified Application Lifecycle Management project.

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

Operating system :	Windows 2000
Test path :	[QualityCenter] Components\YEV\Component\WithDefect

5 Click **Submit** to add the defect information to the Application Lifecycle Management project.

Reference

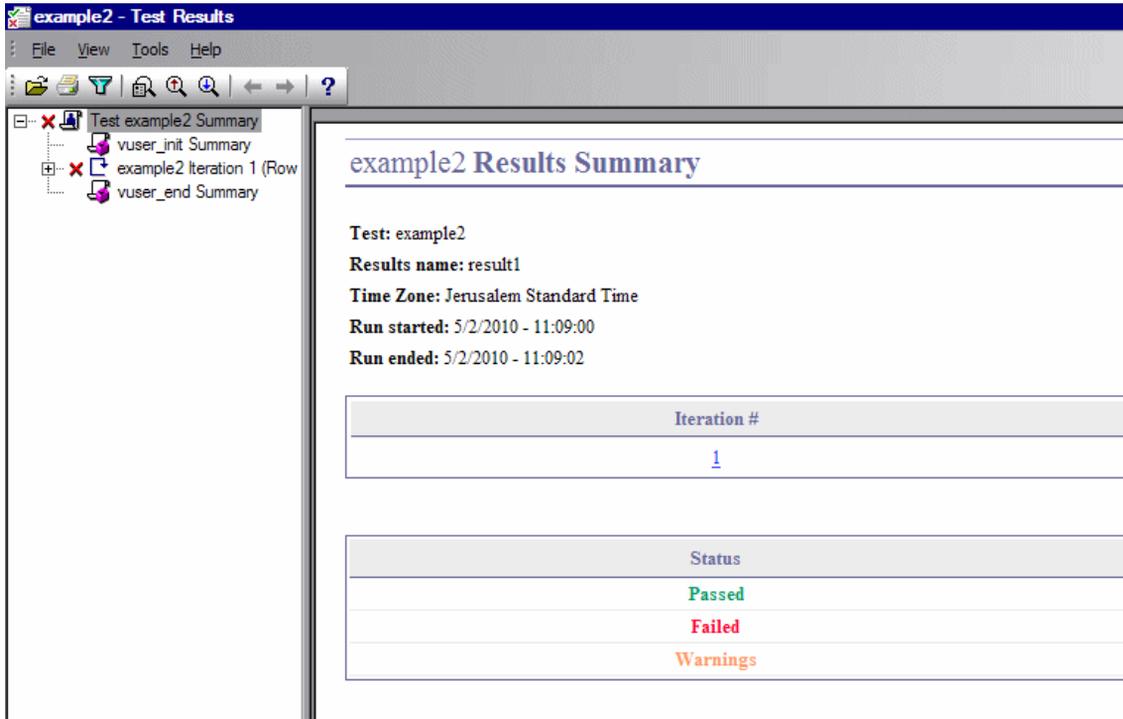
Viewing Test Results User Interface

This section includes:

- Test Results Window on page 174
- Filters Dialog Box on page 177
- Print Dialog Box on page 178
- Print Preview Dialog Box on page 180
- Export to HTML File Dialog Box on page 182

Test Results Window

This window displays a report that summarizes the results of your script run.



To access	Opens automatically after running a script.
Important information	<ul style="list-style-type: none"> ▶ You can configure the display settings from Tools > General Options > Replay > After replay show. ▶ The Test Results window can show results with up to 300 levels in the tree hierarchy. If you have results with more than 300 nested levels, you can view the entire report by manually opening the results.xml file.

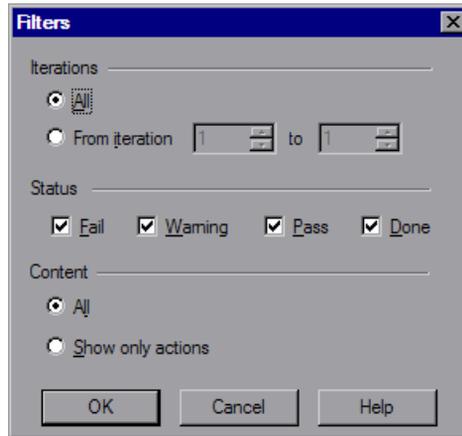
User interface elements are described below:

UI Elements (A-Z)	Description
Report Tree	<p>A graphical representation of the test results located in a pane on the left of the window. You can collapse or expand a branch in the run results tree to change the level of detail that the tree displays.</p> <p>The icons next to the steps indicate the following information:</p> <ul style="list-style-type: none">  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. <p>You can expand and collapse all of the nodes from the View menu, or by clicking *.</p>
Results Details Pain (Results Summary)	<p>Contains details of the test run which change depending on which part of the report tree you select. When you select the top node of the tree, the Result Details tab shows a summary of the results for your test. When you select a branch or step in the tree, the Result Details tab contains the details for that step. The Result Details tab may also include a still image of your application for the highlighted step.</p> <ul style="list-style-type: none"> ➤ Iterations, actions, and steps that contain checkpoints are marked Passed or Failed in the right part of the Test Results window and are identified by icons in the tree pane. ➤ Iterations, actions, and steps that ran successfully, but do not contain checkpoints, are marked Done in the right part of the Test Results window.
Screen Recorder Tab	<p>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.</p>

UI Elements (A-Z)	Description
	Opens the test results of a specific run. For more details, see "How to Open the Test Results of a Specific Run" on page 170.
	Opens the Print dialog box, enabling you to print the test results. For more information, see "Print Dialog Box" on page 178.
	Opens the Filters dialog box, enabling you to filter the test results. For more information, see "Filters Dialog Box" on page 177.
	Opens the Find dialog box, enabling you to search the test results for steps of a particular type. For more information, see "How to Find Steps in the Test Results" on page 171.
	Searches the test results for the previous step matching the criteria in the Find dialog box.
	Searches the test results for the next step matching the criteria specified in the Find dialog box.
	Selects the previous node.
	Selects the next node.
	Opens the product documentation.

Filters Dialog Box

This page enables you to filter the test results in the test results window.



To access	Test Results Window > View > Filters
------------------	--

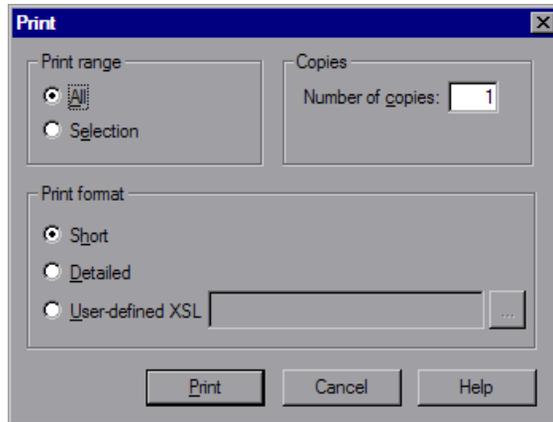
User interface elements are described below:

UI Elements (A-Z)	Description
Iterations	<ul style="list-style-type: none"> ▶ All. Displays test results from all iterations. ▶ From iteration X to Y. Displays the test results from a specified range of test iterations.

UI Elements (A-Z)	Description
Status	<ul style="list-style-type: none"> ➤ 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	<ul style="list-style-type: none"> ➤ All. Displays all steps from all nodes in the test. ➤ Show only actions. Displays the action nodes in the test (not the specific steps in the action nodes).

Print Dialog Box

This dialog box enables you to print the test results.



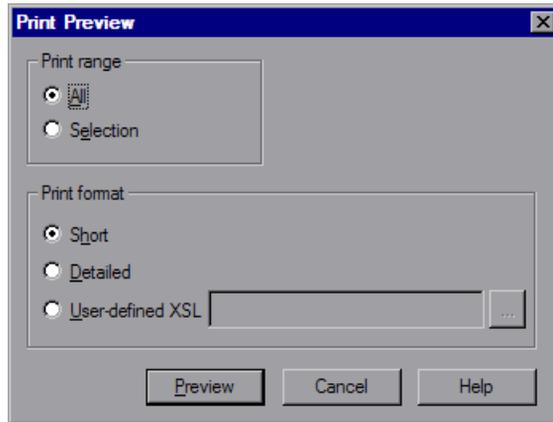
To access	Test Results Window > File > Print
------------------	---

User interface elements are described below:

UI Elements (A-Z)	Description
Print range	<ul style="list-style-type: none"> ▶ All. Prints the results for the entire script. ▶ Selection. Prints the results for the selected branch in the run results tree.
Copies	The number of copies to print.
Print format	<ul style="list-style-type: none"> ▶ Short. Prints a summary line (when available) for each item in the run results tree. This option is only available if you the print range is set to All. ▶ Detailed. Prints all available information for each item in the run results tree, or for the selected branch. ▶ User-defined XSL. Enables you to browse to and select a customized .xsl file. You can create a customized .xsl file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 167.

Print Preview Dialog Box

This dialog box enables you to view a print preview of the test results.



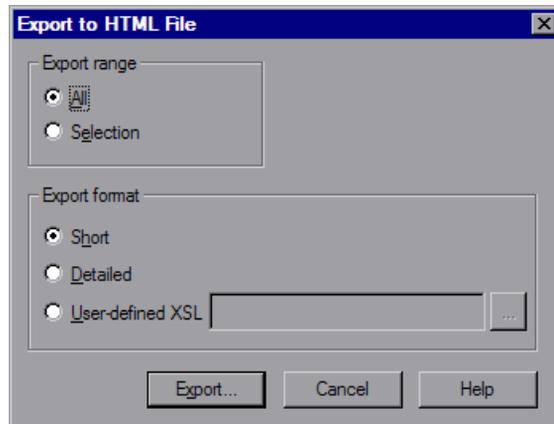
To access	Test Results Window > File > Print Preview
Important information	If some of the information is cut off in the preview, for example, if checkpoint names are too long to fit in the display, click the Page Setup button in the Print Preview window and change the page orientation from Portrait to Landscape .

User interface elements are described below:

UI Elements (A-Z)	Description
Print range	<ul style="list-style-type: none"> ▶ All. Prints the results for the entire script. ▶ Selection. Prints the results for the selected branch in the run results tree.
Print format	<ul style="list-style-type: none"> ▶ Short. Prints a summary line (when available) for each item in the run results tree. This option is only available if you the print range is set to All. ▶ Detailed. Prints all available information for each item in the run results tree, or for the selected branch. ▶ User-defined XSL. Enables you to browse to and select a customized .xsl file. You can create a customized .xsl file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 167.

Export to HTML File Dialog Box

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



To access	Test Results Window > File > Export to HTML File
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
Export Range	<ul style="list-style-type: none"> ▶ All. Exports the results for the entire script. ▶ Selection. Exports result information for the selected branch in the results tree.
Export Format	<ul style="list-style-type: none"> ▶ Short. Prints a summary line (when available) for each item in the run results tree. This option is only available if you the print range is set to All. ▶ Detailed. Prints all available information for each item in the run results tree, or for the selected branch. ▶ User-defined XSL. Enables you to browse to and select a customized .xsl file. You can create a customized .xsl file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 167.

7

Correlation

This chapter includes:

Concepts

- ▶ Correlation Overview on page 187
- ▶ Correlation Vs. Parameterization on page 187
- ▶ Automatic Correlation on page 188
- ▶ Determining Which Values to Correlate on page 188
- ▶ Correlating Java Scripts on page 189
- ▶ Correlating Java Scripts - Serialization on page 192
- ▶ Correlating Winsock Scripts on page 198
- ▶ Wdiff Utility on page 198
- ▶ Modifying Saved Parameters on page 199

Tasks

- ▶ How to Correlate Scripts - Web (HTTP/HTML) on page 200
- ▶ How to Search for Values that Need Correlation on page 201
- ▶ How to Manually Correlate Web Scripts on page 203
- ▶ How to Correlate Scripts - Oracle NCA on page 205
- ▶ How to Correlate Scripts - Database Protocols on page 209
- ▶ How to Correlate Scripts - Microsoft .NET on page 212
- ▶ How to Correlate Scripts - Flex and AMF Protocols on page 215
- ▶ How to Correlate Scripts - Siebel Protocol on page 217
- ▶ How to Correlate Scripts - COM Protocol on page 225

- ▶ How to Correlate Scripts - Tuxedo Protocol on page 227
- ▶ How to Correlate Scripts - Winsock (Tree View) on page 233
- ▶ How to Correlate Scripts - Winsock (Script View) on page 234

Reference

- ▶ Web_reg_save_param function details on page 238
- ▶ Correlation Functions - C Vuser Scripts on page 239
- ▶ Correlation Functions - Java Vuser Scripts on page 241
- ▶ Correlation Functions - Database Vuser Scripts on page 242
- ▶ Correlation User Interface on page 242

Concepts

Correlation Overview

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

For example, 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.

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

Correlation Vs. Parameterization

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

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

Automatic Correlation

Correlation can be performed automatically during recording under certain conditions. If you know the values that will need to be correlated before recording, you can create correlation rules that will automatically correlate those values while you record. Additionally, there are some correlation rules that come pre-defined in VuGen for supported application servers. You can enable or disable rules in the HTTP Correlation node of the Recording Options dialog box. For user interface details, see "HTTP Correlation Node" on page 368.

Determining Which Values to Correlate

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

The following strings most probably require correlation:

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

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

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

Correlating Java Scripts

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.

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);
    }
    :
}
```

Correlating Java Scripts - Serialization

In RMI, and some cases of CORBA, the client AUT creates a new instance of a Java object using the `java.io.Serializable` interface. It passes this instance as a parameter for a server invocation. In the following segment, the instance `p` is created and passed as a parameter.

```
// AUT code:
java.awt.Point p = new java.awt.Point(3,7);
map.set_point(p);
:
```

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user 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. For details, see "Recording Properties Serialization Options Node" on page 403.

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)Ir.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)Ir.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 **Ir.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.

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.

VuGen uses `lrs_save_param` and `lrs_save_searched_string` functions correlate winsock scripts. This means that it stores the data that is received for use in a later point within the 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 correlate. 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 "How to Correlate Scripts - Winsock (Tree View)" on page 233

Wdiff Utility

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

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

For task details, see "How to Search for Values that Need Correlation" on page 201.

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"));
```

Tasks

How to Correlate Scripts - Web (HTTP/HTML)

This task describes the different ways to correlate web (HTTP/HTML) protocol scripts.

This task includes the following steps:

- ▶ "Scan for Correlations" on page 200
- ▶ "Manual Correlation" on page 201
- ▶ "Automatic Correlation Rules" on page 201

Scan for Correlations

- a** Select **Vuser > Scan for Correlations**. This scans the entire script for mismatches between recording and replay snapshot data in web steps. The search occurs in all server responses (including headers).
- b** If mismatches are found they will be displayed in the **Correlation Results** tab of the Output window. For user interface details, see "Output Window - Correlation Results Tab" on page 80.

To determine which values should be correlation see "Determining Which Values to Correlate" on page 188.

- c** When a mismatch is correlated, VuGen adds a **web_reg_save_param_*** function and saves the original value in a comment in the script. Appropriate occurrences of the original value in web steps are replaced with a parameter.
- d** Replay the script. If there are additional correlation-based errors continue to resolve them by repeating this procedure.

If the scan for correlations does not resolve all correlation-based errors, try to resolve them using the manual correlation procedure below.

Manual Correlation

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

a Search for values that need correlation manually. There are a number of ways to manually search for values that need correlation. For details, see "How to Search for Values that Need Correlation" on page 201.

b Correlate the value.

Select one of the following methods:

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

Manually add correlation functions. Manually insert the relevant correlation functions into your script. For details, see "How to Manually Correlate Web Scripts" on page 203.

Automatic Correlation Rules

You can configure VuGen to automatically correlate specific strings by defining correlation rules. For more information, see "HTTP Correlation Node" on page 368.

How to Search for Values that Need Correlation

The following steps describe different ways to search for values that need correlation.

- "Search by Comparing Scripts" on page 201
- "Replay log search" on page 202

Search by Comparing Scripts

- 1** Record a script and save it.
- 2** Create a new script and record the identical operations. Save the script.
- 3** Select **Tools > Compare with Vuser** to compare the scripts. For more details, see "How to Compare Scripts Side by Side" on page 48.

- 4 Differences in the script are highlighted. Review the differences to determine which ones may require correlation.

C:\temp\s2\Action.c	C:\temp\s1\Action.c
<pre> web_submit_data("printpost.asp", "Action=http://testserver/myapplication", "Method=POST", "RecContentType=text/html", "Referer=http://testserver/myapplication", "Snapshot=t5.inf", "Mode=HTML", ITEMDATA, "Name=serial", "Value=123-456-9999", END "Name=ident", "Value=620916F9b8Fc68a58944cb22" LAST); lr_think_time(9); web_link(":123-456-9999:620916F9b8Fc68a58944cb22", "Text=:123-456-9999:620916F9b8Fc68a58944cb22", "Snapshot=t6.inf", LAST); </pre>	<pre> web_submit_data("Action", "Method", "RecCor", "Referer", "Snapshot", "Mode", ITEMDATA, "Name=", "Value=" LAST); lr_think_time(9); web_link(":123-456-9999:620916F9b8Fc68a58944cb22", "Text=", "Snapshot", LAST); </pre>

Note: *WDiff* is the default utility, but you can specify a custom comparison tool. For more information, see "How to Compare Scripts Side by Side" on page 48

Replay log search

- a Scan the script in script view for strings that may need correlation such as hash strings, random strings, session ID's, etc.
- b Search the generation log for the first time that the string appears (this is the response from the server).
- c Search the extended replay log for the same response. Check to see if this response contains a different string within the same boundaries as the original suspected string. If yes, this string requires correlation.

How to Manually Correlate Web Scripts

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

This task includes the following steps:

- ▶ "Locate the string and its details" on page 203
- ▶ "Add `web_reg_save_param_*` function" on page 204
- ▶ "Replace data with parameter" on page 205

1 Locate the string and its details

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

a Identify Patterns using Boundaries

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

- ▶ Analyze the location of the dynamic data within the HTTP response.
- ▶ Identify the string that is immediately to the left of the dynamic data. This string defines the left boundary of the dynamic data.
- ▶ Identify the string that is immediately to the right of the dynamic data. This string defines the right boundary of the dynamic data.
- ▶ The right and left boundaries should be as unique as possible to better locate the strings.
- ▶ `web_reg_save_param_ex` looks for the characters between (but not including) the specified boundaries and saves the information beginning one byte after the left boundary and ending one byte before the right boundary. `web_reg_save_param_ex` does not support embedded boundary characters.

For example, if the input buffer is `{a{b{c}` and `"{"` is specified as a left boundary, and `"}"` as a right boundary, the first instance is `c` and there are no further instances—it found the right and left boundaries but it does not allow embedded boundaries, so `"c"` is the only valid match.

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

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

b Identify Patterns using Xpaths

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

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

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

2 Add web_reg_save_param_* function

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

a web_reg_save_param_ex

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

b web_reg_save_param_xpath

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

3 Replace data with parameter

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

How to Correlate Scripts - Oracle NCA

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

- "Correlate statements for load balancing" on page 205
- "Correlate the icx_ticket variable" on page 207
- "Correlate the JServSessionIdroot values" on page 208

Correlate statements for load balancing

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

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:\apps\ncas...fndnam=apps ");
```

to:

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

The script should now look like this:

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

Correlate the icx_ticket variable

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

```
web_url("fnd_icx_launch.runforms",
  "URL=http://ABC-123:8002/pls/VIS/
  fnd_icx_launch.runforms\?ICX_TICKET=5843A55058947ED3&RESP_APP=AR&RES
  P_KEY=RECEIVABLES_MANAGER&SECGRP_KEY=STANDARD", LAST);
```

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

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

...

web_url("fnd_icx_launch.runforms",
"URL=http://ABC-123:8002/pls/VIS/
fnd_icx_launch.runforms?ICX_TICKET={icx_ticket}&RESP_APP=AR&RESP_KEY=R
ECEIVABLES_MANAGER&SECGRP_KEY=STANDARD", LAST);
```

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

Correlate the JServSessionIdroot values

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

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

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

How to Correlate Scripts - Database Protocols

The following steps describe how to correlate scripts using one of the database protocols.

- "Determine statements that need correlation" on page 209
- "Correlate known values" on page 211

Determine statements that need correlation

If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

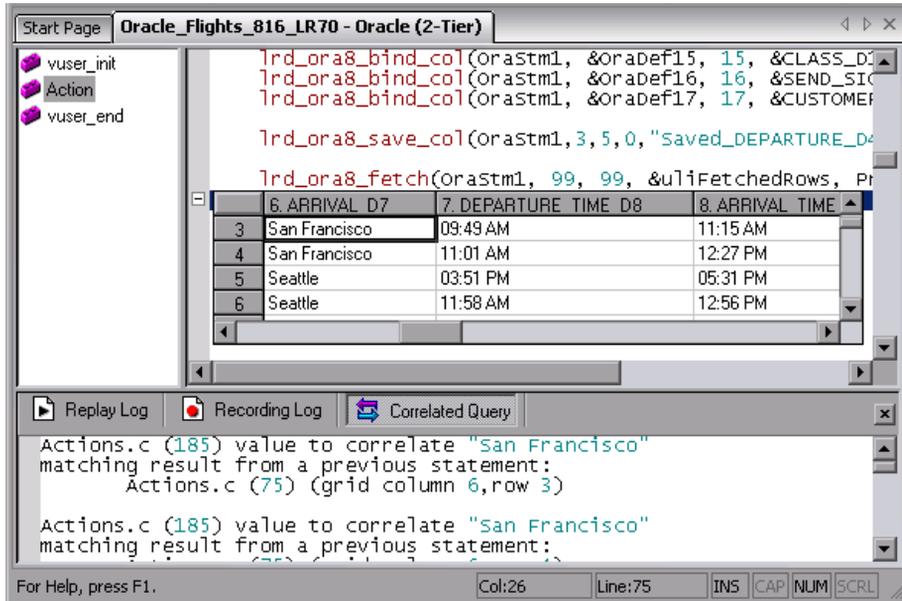
1 Open the Output window.

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.

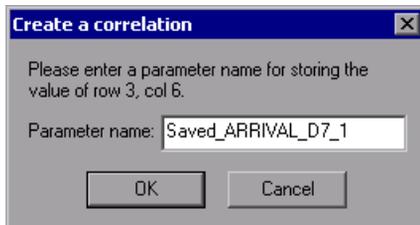
2 Select **Vuser** > **Scan for Correlations**.

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.

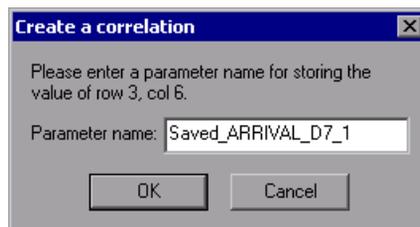
6 Click **Yes** to confirm the correlation.

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.**Correlate known values**

- 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_oras_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).

How to Correlate Scripts - Microsoft .NET

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

This task includes the following steps:

- ▶ "Correlate scripts using ADO.net environments:" on page 213
- ▶ "Correlate with output parameters" on page 214

Correlate scripts using 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 Emparedada	Ana Trujillo	Owner	Avda. de la O
4	ANTON	Antonio Moreno Tacos	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 an **lr.save_string** function before each dataset. For example:

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

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

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

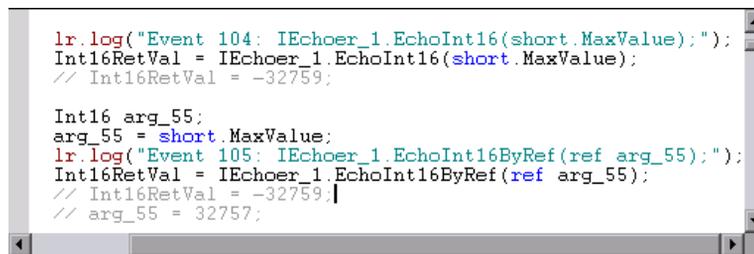
```
lr.message("The customer ID is ""+ lr.eval_string("{MyCustomerID}") + ");
```

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

Correlate with output parameters

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

- 1 Select **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.



```
lr.log("Event 104: IEchoer_1.EchoInt16(short.MaxValue);");
Int16RetVal = IEchoer_1.EchoInt16(short.MaxValue);
// Int16RetVal = -32759;

Int16 arg_55;
arg_55 = short.MaxValue;
lr.log("Event 105: IEchoer_1.EchoInt16ByRef(ref arg_55);");
Int16RetVal = IEchoer_1.EchoInt16ByRef(ref arg_55);
// Int16RetVal = -32759;
// arg_55 = 32757;
```

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

How to Correlate Scripts - Flex and AMF Protocols

This task describes how to correlate flex and AMF protocol scripts.

To correlate flex and AMF scripts:

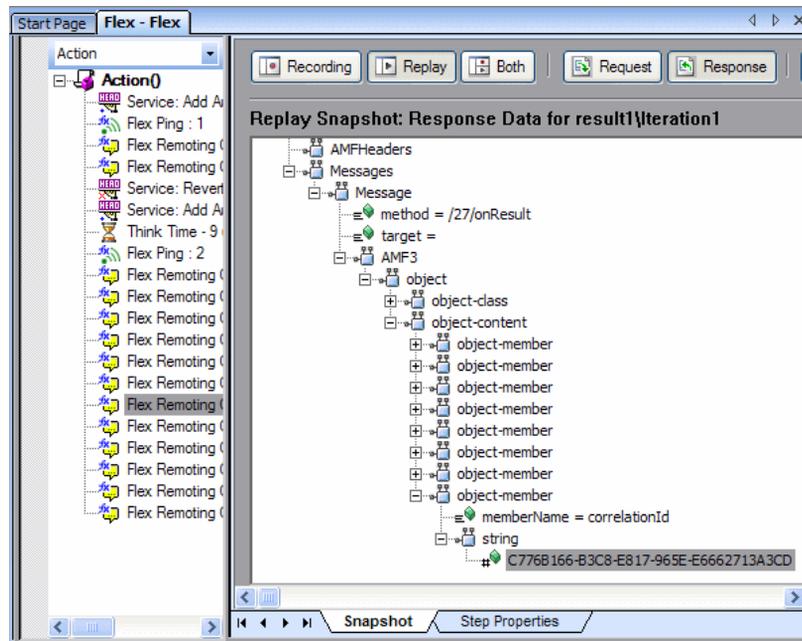
1 Locate the step in your script that failed due to dynamic values that need correlation.

Use the Replay Log to assist you in finding the problematic step. These errors are not always obvious, and you may only detect them by carefully examining Vuser log files.

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.



The screenshot displays a software interface with a window titled "Flex - Flex". On the left, there is a tree view under "Action()" containing a list of steps: "Service: Add Av", "Flex Ping : 1", "Flex Remoting C", "Service: Revert", "Service: Add Av", "Think Time - 9", "Flex Ping : 2", and several "Flex Remoting C" entries. The main area on the right is titled "Replay Snapshot: Response Data for result1\iteration1" and shows a hierarchical tree structure of response data. The structure includes "AMFHeaders", "Messages", and "Message". Under "Message", there is a "method = /27/onResult" and a "target =" field. Below "target =", there is an "AMF3" object containing an "object" with "object-class", "object-content", and multiple "object-member" entries. One "object-member" has a "memberName = correlationId" and a "string" value of "C776B166-B3C8-E817-965E-E6662713A3CD". At the bottom of the window, there are navigation buttons and tabs labeled "Snapshot" and "Step Properties".

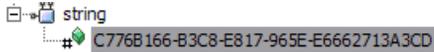
3 Save the entire server response to a parameter.

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 (or AMF) 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 XML tree of the Server Response, right-click the node above the value (for example, string), and select **Save value in parameter**.



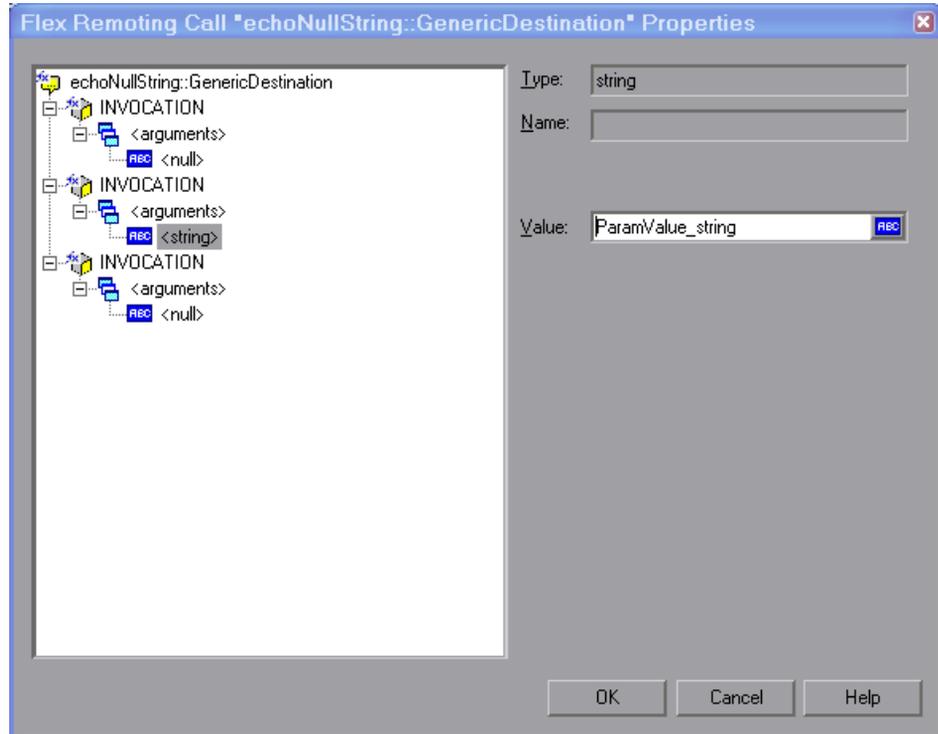
- In the XML Parameter Properties dialog, specify a parameter **Name**. You will use this name in subsequent steps.
- Click **OK**. The script will now contain a new function, **lr_xml_get_values**.

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.

How to Correlate Scripts - Siebel Protocol

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

- "Correlation library" on page 218
- "Correlation Rules" on page 218
- "Correlate SWECCount parameters" on page 224

- ▶ "Correlate ROWID parameters" on page 224
- ▶ "Correlate SWET (timestamp) parameters" on page 225

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 > HTTP Properties > Advanced** node.
- 4** Open the recording option's Correlation node and click **Import**. Import the rules file, **WebSiebel77Correlation.cor**, from the \dat\webrulesdefaultsetting directory. If you are prompted with warnings, click **Override**.

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.

Correlation Rules

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

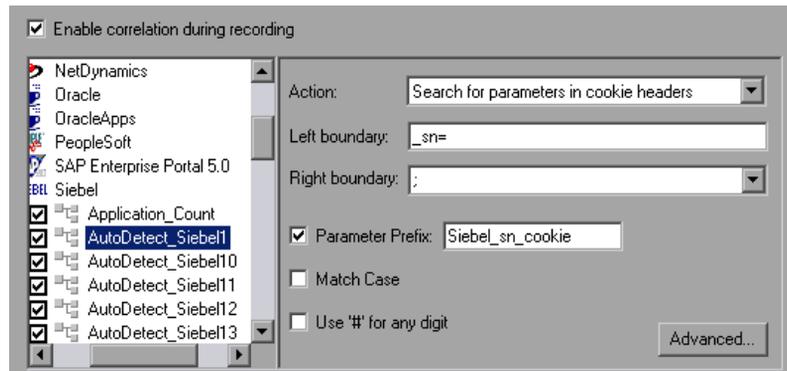
When VuGen detects a match using the boundary criteria, it saves the value between the boundaries to a parameter. The value can be a simple variable or a public function.

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

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

Simple Variable Correlation

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



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

```

/* Registering parameter(s) from source
web_reg_save_param("Siebel_sn_cookie6",
"LB/IC=_sn=",
"RB/IC=;",
"Ord=1",
"Search=headers",
"RelFrameId=1",
LAST);

...

web_url("start.swe_3",
"URL=http://cannon.hplab.com/callcenter_enu/
start.swe?SWECmd=GotoPostedAction&SWEDIC=true&_sn={Siebel_sn_cookie6}&S
WEC={Siebel_SWECCount}&SWEFrame=top._sweclient&SWECS=true",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://cannon.hplab.com/callcenter_enu/
start.swe?SWECmd=GetCachedFrame&_sn={Siebel_sn_cookie6}&SWEC={Siebel_S
WECCount}&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 222.

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-mOTFXHWBAAGb5Xzv9Ls2Z45QvxG
QnOnPVtX6vnfUU_&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-mOTFXHWBAAGb5Xzv9Ls2Z45QvxGQnOnPVtX6vnf
UU_&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_SWECCount_var`). Before each step, VuGen saves the counter's value to a VuGen parameter (`Siebel_SWECCount`).

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

```
Siebel_SWECCount_var += 1;

lr_save_int(Siebel_SWECCount_var, "Siebel_SWECCount");

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

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

Correlate SWECCount parameters

The SWECCount 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 SWECCount correlations is the same.

Correlate ROWID parameters

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

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

If you define parameters to correlate the string as

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

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

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

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

Correlate SWET (timestamp) parameters

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

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

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

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

How to Correlate Scripts - COM Protocol

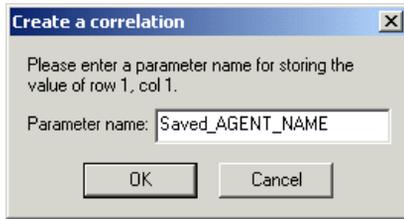
The following steps describe how to correlate COM protocol scripts.

- "Find values the need correlation" on page 225
- "Correlate known values" on page 226

Find values the need 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.

Correlate known values

- 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.

How to Correlate Scripts - Tuxedo Protocol

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*.

- "Determine values that need correlation" on page 228
- "Correlate for FML/FML32 buffers" on page 229
- "Correlate based on buffer location" on page 230
- "Correlate based on delimiter" on page 232

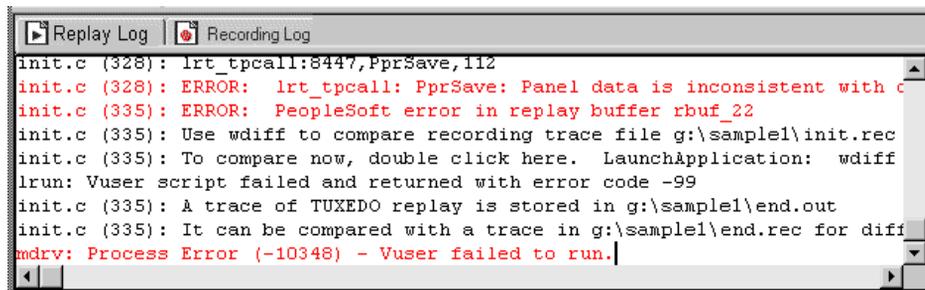
Determine values that need correlation

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

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.**



```

Replay Log | Recording Log
init.c (328): lrt_tpcall:8447,PprSave,112
init.c (328): ERROR: lrt_tpcall: PprSave: Panel data is inconsistent with c
init.c (335): ERROR: PeopleSoft error in replay buffer rbuf_22
init.c (335): Use wdiff to compare recording trace file g:\sample1\init.rec
init.c (335): To compare now, double click here. Launch&application: wdiff
lrn: Vuser script failed and returned with error code -99
init.c (335): A trace of TUXEDO replay is stored in g:\sample1\end.out
init.c (335): It can be compared with a trace in g:\sample1\end.rec for diff
mdrv: Process Error (-10348) - Vuser failed to run.
  
```

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

For more information, see "Wdiff Utility" on page 198.

Correlate for FML/FML32 buffers

Use `lrt_save_fld_val` or `lrt_save32_fld_val` to save the contents of the FML or FML32 buffer.

To correlate statements using `lrt_save_fld_val`:

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

Example:

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

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

Example:

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

```
/* Fill the data_0 buffer with new account information*/
data_0 = lrt_tmalloc("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*/
tresult_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_fid((FBFR*)data_0, "name=ACCOUNT_ID", "value={account_id}",
LRT_END_OF_PARAMS);
lrt_Fadd_fid((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_fid_val((FBFR*)data_0, "id=8302", 0, "account_id");
```

Correlate based on buffer location

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

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

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

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

Example:

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

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

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

```
/* Request CARRAY buffer 57 */
  lrt_memcpy(data_0, buf_143, 227);
  tresult_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\x0\x0"
  "SPprLoadReq"
  "\xff\x0\x10\x0\x0\x4\x3\x6"
  "{empid}" // G001
  "\x7"
  "Claudia"
  "\xe"
  "LAST_NAME_SRCH"
...
```

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

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

Correlate based on delimiter

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

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

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

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

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

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

Example:

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

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

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

```

/* Request CARRAY buffer 1 */
lrt_memcpy(data_0, sbuf_1, 41);
lrt_display_buffer("sbuf_1", data_0, 41, 41);
data_1 = lrt_tmalloc("CARRAY", "", 8192);
tresult_int = lrt_tpcall("GetCertificate",
    data_0,
    41,
    &data_1,
    &olen,
    TPSIGRSTRRT);

/* Reply CARRAY buffer 1 */
lrt_display_buffer("rbuf_1", data_1, olen, 51);
lrt_abort_on_error();

lrt_save_searched_string(data_1, olen, 0, "SCertRep", 9, 16, "cert1");

```

How to Correlate Scripts - Winsock (Tree View)

This task describes how to create a correlation in a Winsock script from the Tree View.

This task includes the following steps:

- "Select the desired text to correlate" on page 234
- "Modify the script statement - optional" on page 234
- "Create the parameter and replace instances" on page 234

1 Select the desired text to correlate

In the snapshot window, right-click and select Create Parameter. Specify the boundaries of the parameter according to the dialog box. For user interface details, see "Create Parameter Dialog Box" on page 243.

2 Modify the script statement - optional

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**).

3 Create the parameter and replace instances

Click **OK** to create the parameter. VuGen asks you for a confirmation before replacing the parameter. Click **Yes** to replace all instances in the send buffers after this occurrence with the parameter. Click **No** to generate a list of all the instances without replacing them with the parameter.

4 Manage the parameter instances

You can manage all the instances of a parameter in the **Parameters** tab of the **Output** window. For user interface details, see "Parameter Tab" on page 97.

How to Correlate Scripts - Winsock (Script View)

This task describes how to create a correlation in a Winsock script manually from the Script View.

To correlate 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** View the buffer contents by selecting **data.ws** in the Action Pane of the main VuGen window (displayed by default in Script View). Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in parameter braces. The default parameter braces are brackets (< > or ()). You can modify the parameter braces in the **Tools > General Options > Parameterization tab**.

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

```

frodo:/u/jay>ps
  PID TTY   TIME CMD
14602 pts/18 0:00 clock
14569 pts/18 0:03 tcsh

frodo:/u/jay>kill 14602
[3] Exit 1          clock
frodo:/u/jay>

```

During execution, the PID of the procedure is different (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"

```


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

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

Reference

Web_reg_save_param function details

When you run a script, the **web_reg_save_param** function scans the subsequent HTML page that is accessed. You specify a left and/or right boundary and VuGen searches for text between those boundaries. When VuGen finds the text, it assigns it to a parameter.

The function's syntax is as follows:

```
int web_reg_save_param (const char *mpszParamName, <List of Attributes>, LAST);
```

The following table lists the available attributes. Note that the attribute value strings (for example, Search=all) are not case sensitive.

NotFound	The handling method when a boundary is not found and an empty string is generated. "ERROR," the default, indicates that VuGen should issue an error when a boundary is not found. When set to "EMPTY," no error message is issued and script execution continues. Note that if Continue on Error is enabled for the script, then even when NOTFOUND is set to "ERROR," the script continues when the boundary is not found, but it writes an error message to the Extended log file.
LB	The left boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/" BIN" after the boundary to specify binary data.
RB	The right boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/" BIN" after the boundary to specify binary data.

RelFrameID	The hierarchy level of the HTML page relative to the requested URL. The possible values are ALL or a number.
Search	The scope of the search—where to search for the delimited data. The possible values are Headers (search only the headers), Body (search only Body data, not headers), or ALL (search Body and headers). The default value is ALL.
ORD	This optional parameter indicates the ordinal or occurrence number of the match. The default ordinal is 1. If you specify "All," it saves the parameter values in an array.
SaveOffset	The offset of a sub-string of the found value, to save to the parameter. The default is 0. The offset value must be non-negative.
Savelen	The length of a sub-string of the found value, from the specified offset, to save to the parameter. The default is -1, indicating until the end of the string.
Convert	The conversion method to apply to the data: HTML_TO_URL: convert HTML-encoded data to a URL-encoded data format HTML_TO_TEXT: convert HTML-encoded data to plain text format

Correlation Functions - C Vuser Scripts

To correlate statements for protocols that do not have specific functions, you can use the C Vuser correlation functions. These functions can be used for all C-type Vusers, to save a string to a parameter and retrieve it when required.

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

For additional information about the syntax of these functions, see the *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");
```

Correlation Functions - Java Vuser Scripts

To correlate statements for Java Vusers, you can use the Java Vuser correlation functions. These functions may be used for all Java type Vusers, to save a string to a parameter and retrieve it when required.

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

When recording a CORBA or RMI session, VuGen performs correlation internally. For more information, see "Correlating Java Scripts" on page 189

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
```

Correlation Functions - Database Vuser Scripts

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 *Online Function Reference* (**Help > Function Reference**).

Correlation User Interface

This section includes:

- ▶ Create Parameter Dialog Box on page 243

Create Parameter Dialog Box

This dialog box enables you to correlate data and create parameters in Winsock scripts.

To access	Tree View > Right-click menu > Create Parameter
-----------	---

User interface elements are described below:

UI Elements (A-Z)	Description
Parameter Name	The name of the parameter.
Data Range	You can define the parameter by a start and end range in bytes. Enter the numbers manually in the From and To fields or click Select Range and highlight the desired text.
Boundaries	You can define the parameter by defining a left and right boundary. To do so, select Extract parameter data using boundaries . Click the button to right of the Left field, highlight the desired text, and click Done . Repeat the procedure for the Right boundary.
Script Statement	The statement that will appear in your script based on the options selected in this dialog box. You can manually edit this statement.

8

Working with Application Lifecycle Management

This chapter includes:

Concepts

- ▶ Managing Scripts Using ALM Overview on page 246
- ▶ ALM Version Control Overview on page 246

Tasks

- ▶ How to Work with Scripts in ALM Projects on page 247
- ▶ How to Work with Version Controlled Scripts in ALM Projects on page 248
- ▶ How to Save VuGen Scripts to ALM Projects on page 249
- ▶ How to View/Modify Previous Versions of a Script on page 250

Reference

- ▶ ALM User Interface on page 252

Concepts

Managing Scripts Using ALM Overview

VuGen works together with HP Application Lifecycle Management (ALM). ALM provides an efficient method for storing and retrieving Vuser scripts, scenarios, and results. You can store scripts in an ALM project and organize them into unique groups.

In order for VuGen to access an ALM project, you must connect it to the Web server on which ALM is installed. You can connect to either a local or remote Web server.

For more information on working with ALM, see the *Application Lifecycle Management User Guide*.

ALM Version Control Overview

VuGen supports version control features in scripts saved in ALM projects that use version control and have the Performance Center addition installed.

The version control features change the process of opening and saving a script. Scripts with version control are either in a state of checked-in or checked-out. When you are working with a script in a checked-out state, any changes you make will not be saved on the ALM server until you check in the script. If you save the script from within VuGen, a temporary file is saved onto your machine that protects your changes in case your computer crashes.

If you are working with a script in a checked-in state, the script is read-only and you cannot save your changes until you check out the script.

If a particular script is being saved to ALM for the first time, and the project uses version control, the script automatically starts in a checked-out state.

Tasks

How to Work with Scripts in ALM Projects

The following steps describe the workflow of how to work with scripts saved in an ALM project.

Note: To work with scripts in ALM projects with version control, see "How to Work with Version Controlled Scripts in ALM Projects" on page 248.

- "Connect to ALM" on page 247
- "Open the script" on page 247
- "Save the script" on page 247

Connect to ALM

Open a connection to the ALM server and project that contains the script. For task details, see "Connect to ALM" on page 248.

Open the script

Select **File > Open** and specify the location of the script.

Save the script

Select **File > Save**. If the script is in a project that uses version control and is not checked out, the script is only saved as a temporary file on your local machine.

Connect to ALM

To store and retrieve scripts from ALM, you need to connect to an ALM project. You can connect or disconnect from an ALM project at any time during the testing process.

The connection process has two stages. First, you connect to an ALM Web server. This server handles the connections between VuGen and the ALM project.

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

To connect to ALM:

- 1** Select **Tools > HP ALM Connection**.
- 2** Complete the HP ALM Connection dialog box and select **Connect**.
- 3** Complete the remainder of the HP ALM Connection dialog box. For user interface details, see "HP ALM Connection Dialog Box" on page 253 .
- 4** To disconnect from ALM, click **Disconnect**.

How to Work with Version Controlled Scripts in ALM Projects

The following steps describe the workflow of how to work with scripts saved in ALM projects that use version control.

Note: This procedure is only relevant for scripts in ALM projects that support version control and have the Performance Center addition installed. If these two conditions are not met, see "How to Work with Scripts in ALM Projects" on page 247.

- "Connect to ALM" on page 249
- "Open the script" on page 249
- "Check in/out the script" on page 249

- "Cancel a check out (optional)" on page 249
- "Save the script" on page 249

Connect to ALM

Open a connection to the ALM server and project that contains the script. For task details, see "Connect to ALM" on page 248.

Open the script

Select **File > Open** and specify the location of the script.

Check in/out the script

If the ALM project has version control, each script is always defined as being either checked-in or checked-out. For more details, see "ALM Version Control Overview" on page 246. To check in and check out scripts, select **File > HP ALM Version Control > Check In/Out**.

Cancel a check out (optional)

If you checked out a script and do not want to save the changes, you can return the status of the script to checked-in without saving by selecting **File > HP ALM Version Control > Undo Check Out**.

Save the script

Select **File > Save**. If the script is in a project that uses version control and is not checked out, the script is only saved as a temporary file on your local machine.

How to Save VuGen Scripts to ALM Projects

The following steps describe how to save a VuGen script to an ALM project.

- "Open/create the VuGen script" on page 250
- "Connect to ALM" on page 250
- "Save the script to ALM" on page 250

Open/create the VuGen script

Create or open the desired script in VuGen.

Connect to ALM

Open a connection to the ALM server and project that you want to store the script. For task details, see "Connect to ALM" on page 248.

Save the script to ALM

Select **File > Save as** and specify the location.

How to View/Modify Previous Versions of a Script

If your script is saved in an ALM project that uses version control, you can view, modify, and save previous versions of the script. The following steps describe how to do this.

- "Connect to ALM" on page 250
- "Open the script" on page 250
- "View a previous version of a script" on page 250
- "Check out a previous version of a script" on page 251

Connect to ALM

Open a connection to the ALM server and project that you want to store the script. For task details, see "Connect to ALM" on page 248.

Open the script

Select **File > Open** and specify the location.

View a previous version of a script

To view a previous version of the script in read-only mode, select **File > HP ALM Version Control > Version History** and click **Get Version**.

Check out a previous version of a script

Select **File > ALM Version Control > Version History** and click **Check Out**. To check this version in as the new version, select **File > HP ALM Version Control > Check In**. To return the script without saving select **File > HP ALM Version Control > Undo Check Out**.

Reference

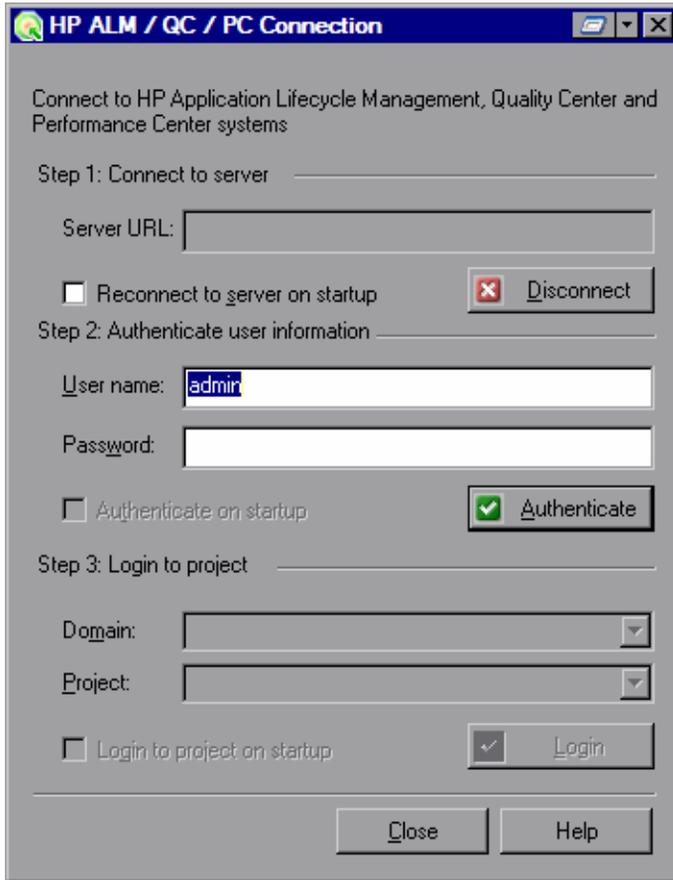
ALM User Interface

This section includes:

- ▶ HP ALM Connection Dialog Box on page 253
- ▶ Upload Script Dialog Box on page 256

HP ALM Connection Dialog Box

This dialog box enables you to connect to an ALM project from within VuGen.



HP ALM / QC / PC Connection

Connect to HP Application Lifecycle Management, Quality Center and Performance Center systems

Step 1: Connect to server

Server URL:

Reconnect to server on startup

Step 2: Authenticate user information

User name:

Password:

Authenticate on startup

Step 3: Login to project

Domain:

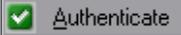
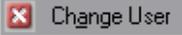
Project:

Login to project on startup

To access	Tools > HP ALM Connection > Connect
-----------	-------------------------------------

User interface elements are described below:

UI Elements (A-Z)	Description
Step 1: Connect to Server	<ul style="list-style-type: none">➤ Server URL. The URL of the server that contains ALM.➤ Reconnect to server on startup. Automatically reconnect to the server every time you start the application.➤  Connect /  Disconnect. Connects to the server specified in the Server URL box. Only one button is visible at a time, depending on your connection status.

UI Elements (A-Z)	Description
<p>Step 2: Authenticate User Information</p>	<ul style="list-style-type: none"> ➤ User Name. Your ALM project user name. ➤ Password. Your ALM project password. ➤ Authenticate on startup. Authenticates your user information automatically, the next time you open the application. This option is only available if you selected Reconnect to server on startup above. ➤  Authenticate . Authenticates your user information against the ALM 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  Change User . You can log in to the same ALM server using a different user name by clicking Change User, entering a new user name and password, and then clicking Authenticate again.
<p>Step 3: Login to Project</p>	<ul style="list-style-type: none"> ➤ Domain. The domain that contains the ALM 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.) ➤ Project. Enter the ALM project name or select a project from the list. Only those projects that you have permission to connect to are displayed. ➤ Login to project on startup. This option is only enabled when the Authenticate on startup check box is selected. ➤  Login /  Logout .Logs into and out of the ALM project.

 **Upload Script Dialog Box**

This dialog box enables you to open a script from an ALM project or save a script to an ALM project.

To access	Connect to ALM > Save script created in VuGen to ALM
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
Upload run time files	Uploads only the files needed to replay the script. Does not upload recording snapshot files and other unnecessary files. This results in a shorter download time.
Upload all files	Uploads all of the files associated with this script. This results in a longer upload time.

9

Parameters

This chapter includes:

Concepts

- ▶ Parameter Overview on page 258
- ▶ Parameter Types on page 260
- ▶ Data Assignment Methods for File/Table/XML Parameters on page 263
- ▶ Tuxedo and PeopleSoft Parameters on page 268
- ▶ XML Parameters on page 268

Tasks

- ▶ How to Create a Parameter on page 277
- ▶ How to Create an XML Parameter from a Web Service Call on page 279
- ▶ How to Work with Existing Parameters on page 280
- ▶ How to Import Parameter Data from a Database on page 280

Reference

- ▶ Parameter User Interface on page 282

Troubleshooting and Limitations on page 304

Concepts

Parameter Overview

When you record a business process, VuGen generates a script that contains the actual values used during recording. Suppose you want to perform the script's actions (query, submit, and so forth) using different values from those recorded. To do this, you replace the recorded values with parameters. This is known as *parameterizing* the script.

The resulting Vusers substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables.

You can use parameterization only for the arguments within a function. You cannot parameterize text strings that are not function arguments. In addition, not all function arguments can be parameterized. For details on which arguments you can parameterize, see the *Online Function Reference (Help > Function Reference)* for each function.

Input parameters are parameters whose value you define in the design stage before running the script. Output parameters you define during design stage, but they acquire values during test execution. Output parameters are often used with Web Service calls.

Use care when selecting a parameter for your script during design stage, make sure that it is not an empty Output parameter.

Example:

Let's say you recorded a Vuser script while operating a Web application. VuGen generated the following statement that searches a library's database for the title "UNIX":

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

When you replay the script using multiple Vusers and iterations, you do not want to repeatedly use the same value, UNIX. Instead, you replace the constant value with a parameter:

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

Parameter Types

Every parameter is defined by the type of data it contains. This section contains information on the different parameter types.

File Parameter Types

Data files hold data that a Vuser accesses during script execution. Data files can be local or global. You can specify an existing ASCII file, use VuGen to create a new one, or import a database file. Data files are useful if you have many known values for your parameter.

The data in a data file is stored in the form of a table. One file can contain values for many parameters. Each column holds the data for one parameter. Column breaks are marked by a delimiter, for example, a comma.

In the following example, the data file contains ID numbers and first names:

```
id,first_name
120,John
121,Bill
122,Tom
```

Note: When working with languages other than English, save the parameter file as a UTF-8 file. In the Parameter Properties window, click **Edit with Notepad**. In Notepad, save the file as a text file with UTF-8 type encoding.

Table Parameter Types

The Table parameter type is meant for applications that you want to test by filling in table cell values. Whereas the file type uses one cell value for each parameter occurrence, the table type uses several rows and columns as parameter values, similar to an array of values. Using the table type, you can fill in an entire table with a single command. This is common in SAPGUI Vusers where the `sapgui_table_fill_data` function fills the table cells.

XML Parameter Types

Used as a placeholder for multiple valued data contained in an XML structure. You can use an XML type parameter to replace the entire structure with a single parameter. For example, an XML parameter called **Address** can replace a contact name, an address, city, and postal code. Using XML parameters for this type of data allows for cleaner input of the data, and enables cleaner parameterization of Vuser scripts. We recommend that you use XML parameters with Web Service scripts or for SOA services.

Internal Data Parameter Types

Internal data is generated automatically while a Vuser runs, such as Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.

- ▶ **Date/Time:** The current date/time. You can specify the format and the offset in the Parameter Properties dialog box.
- ▶ **Group Name:** The name of the Vuser Group. If there is no Vuser Group (for example, when running a script from VuGen) the value is always **none**.
- ▶ **Iteration Number:** The current iteration number.
- ▶ **Load Generator Name:** The name of the Vuser script's load generator (the computer on which the Vuser is running).
- ▶ **Random Number:** A random number within a range of values that you specify.

- ▶ **Unique Number:** Assigns a range of numbers to be used for each Vuser. You specify the start value and the block size (the amount of unique numbers to set aside for each Vuser). For example, if you specify a start value of 1 and a block size of 100 the first Vuser can use the numbers 1 to 100, the second Vuser can use the numbers 201-300, etc.
- ▶ **Vuser ID:** The ID number assigned to the Vuser by the Controller during a scenario run. When you run a script from VuGen, the Vuser ID is always -1.

Note: This is not the ID number that appears in the Vuser window—it is a unique ID number generated at runtime.

User-Defined Function Parameters

Data that is generated using a function from an external DLL. A user-defined function replaces the parameter with a value returned from a function located in an external DLL.

Before you assign a user-defined function as a parameter, you create the external library (DLL) with the function. The function should have the following format:

```
__declspec(dllexport) char *<functionName>(char *, char *)
```

The arguments sent to this function are both NULL.

When you create the library, we recommend that you use the default dynamic library path. That way, you do not have to enter a full path name for the library, but rather, just the library name. VuGen's bin 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 = tunelessly); static char t[35]; strcpy(t, ctime( &x)); t[24] = '\0'; return t;}
```

BPT Type Parameters

You typically use BPT (Business Process Test) type parameters to share parameters between business components in Application Lifecycle Management. In the Parameter Properties dialog box, you can configure properties such as Input/Output, value, data type and description. For more information, see "Parameter Properties Dialog Box" on page 284.

Note: BPT type parameters are only available with an HP Service Test license. For details, contact HP Support.

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

Data Assignment Methods for File/Table/XML Parameters

When using values from a file, VuGen lets you specify the way in which you assign data from the source to the parameters. The following methods are available:

- Sequential
- Random
- Unique

Sequential

Assigns data to a Vuser sequentially. As a running Vuser accesses the data table, it takes the next available row of data.

If there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random

Assigns a random value from the data table every time a new parameter value is requested.

When running a scenario in LoadRunner, or a script in HP Business Process Monitor, you can specify a seed number for random sequencing. Each seed value represents one sequence of random values used for test execution. Whenever you use this seed value, the same sequence of values is assigned to the Vusers in the scenario. You enable this option if you discover a problem in the test execution and want to repeat the test using the same sequence of random values.

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

Unique

Assigns a unique sequential value to the parameter for each Vuser. Ensure that there is enough data in the table for all Vusers and their iterations. If you have 20 Vusers and you want to perform 5 iterations, your table must contain at least 100 unique values.

If you run out of unique values, VuGen behaves according to the option you select in the **When out of values** field. For more information, see "File parameters" on page 287.

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 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 Data Assignment method is determined by the **Select next row** field, and the Update method is determined by the **Update value on** field.

The following table summarizes the values that Vusers use depending on which Data Assignment and Update properties you selected:

Update Method	Data Assignment Method		
	Sequential	Random	Unique
Each iteration	The Vuser takes the <i>next</i> value from the data table for each iteration.	The Vuser takes a <i>new random</i> 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 <i>next</i> value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a <i>new random</i> value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a <i>new unique</i> value from the data table for each occurrence of the parameter, even if it is within the same iteration.
Once	The value assigned in the first iteration is used for all subsequent iterations for each Vuser.	The random value assigned in the first iteration is used for all iterations of that Vuser.	The unique value assigned in the first iteration is used for all subsequent iterations of the Vuser.

Examples

Assume that your table/file has the following values:

Kim; David; Michael; Jane; Ron; Alice; Ken; Julie; Fred

Sequential Method

- If you specify update on **Each iteration**, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, and so on.
- If you specify update on **Each occurrence**, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, and so on.
- If you specify update **Once**, all Vusers take Kim for all iterations.

Note: If you select the **Sequential** method and there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random Method

- If you specify update on **Each iteration**, the Vusers use random values from the table for each iteration.
- If you specify update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
- If you specify update **Once**, all Vusers take the first randomly assigned value for all the iterations.

Unique Method

- If you specify update on **Each iteration**, for a test run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane, Ron, and Alice. The third Vuser, Ken, Julie, and Fred.
- If you specify update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.

- If you specify update **Once**, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, and so on.

Vuser Behavior in the Controller (LoadRunner Only)

When you set up a scenario to run a parameterized script, you can instruct the Vusers how to act when there are not enough values. The following table summarizes the results of a scenario using the following parameter settings:

- Select next row = **Unique**
- Update Value on = **Each iteration**
- When out of values = **Continue with last value**

Situation	Duration	Resulting Action
More iterations than values	Run until completion	When the unique values are finished, each Vuser continues with the last value, but a warning message is sent to the log indicating that the values are no longer unique.
More Vusers than values	Run indefinitely or Run for ...	Vusers take all of the unique values until they are finished. Then the test issues an error message Error: Insufficient records for param <param_name> in table to provide the Vuser with unique data. To avoid this, change the When out of values option in the Parameter properties or the Select next row method in the Parameter properties.
One of two parameters are out of values	Run indefinitely or Run for ...	The parameter that ran out of values, continues in a cyclic manner until the values of the second parameter are no longer unique.

Tuxedo and PeopleSoft Parameters

Tuxedo scripts contain strings of type "name=..." or "value=...". You can only define parameters for the portion of the string following the equal sign (=). For example:

```
lrt_Fadd_fid((FBFR*)data_0,"name=PHONE","value={parameter_1}",
LRT_END_OF_PARMS);
```

In general, we recommend that you use **lrt_save_parm** to save a portion of a character array to a parameter. Use **lrt_save_searched_string** when you want to save information, relative to the position of a particular string in a character array. For PeopleSoft Vusers, we recommend that you use **lrt_save_searched_string**, since the reply buffers returned from the PeopleSoft server often differ in size during replay from what was seen during recording.

XML Parameters

When you create a Web Service call to emulate a specific operation, the arguments in the operation may include complex structures with many values. You can use an XML type parameter to replace the entire structure with a single parameter.

You can create several value sets for the XML elements and assign a different value set for each iteration.

The XML parameter type supports complex schema types such as arrays, Choice, and <any> elements.

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.

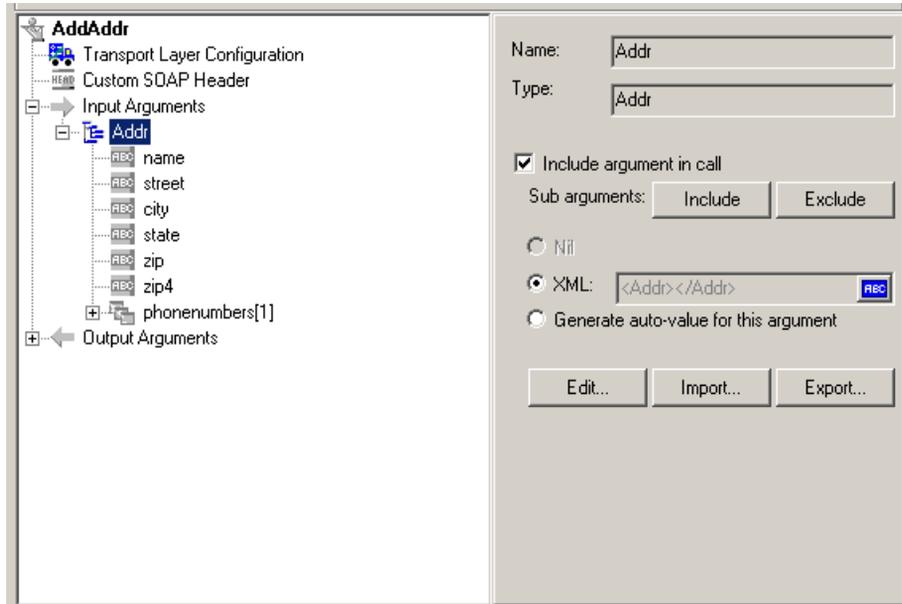
You can create new XML type parameters directly from the **Insert** menu, similar to all other parameter types. For Web Services type scripts, you create an XML parameter directly from the Web Services Call properties.

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.

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.

Schema	Set 1	Set 2	Set 3
[-] Addr			
<input checked="" type="checkbox"/> name	John Doe	Tom Smith	Kim Jones
<input checked="" type="checkbox"/> street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
<input checked="" type="checkbox"/> city	Delray Beach	NIL	NIL
<input checked="" type="checkbox"/> state	FL	AZ	MA
<input checked="" type="checkbox"/> zip	33452	NIL	02134
<input checked="" type="checkbox"/> zip4			
[-] phonenumber			
[-] PhoneNumber [...]			
[-] PhoneNumber[1]	NIL	NIL	NIL

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			
[ABC] name	John Doe	Tom Smith	Kim Jones
[ABC] street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
[ABC] city	Delray Beach	NIL	NIL
[ABC] state	FL	AZ	MA
[ABC] zip	33452	NIL	02134
[ABC] zip4			
[-] phonenumber			
[-] PhoneNumber [..]			
[-] PhoneNumber[1]	NIL	NIL	NIL

To set parameter element values:

1 View the Parameter Properties.

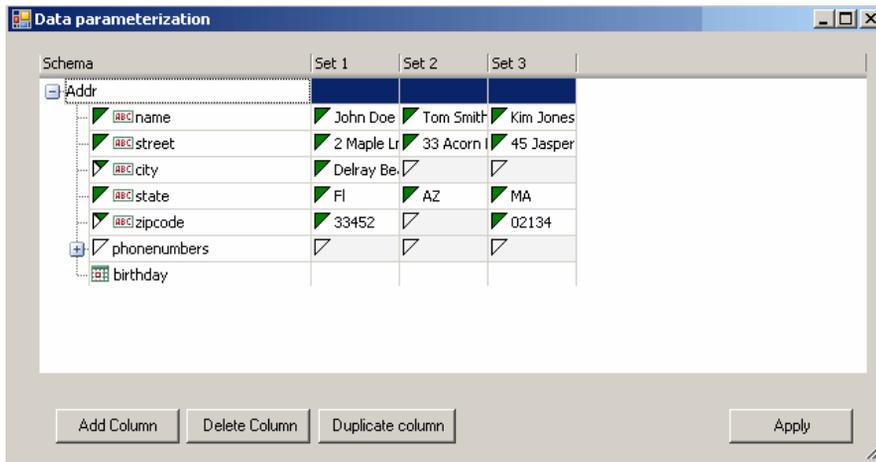
If the Parameter Properties dialog box is not open, select **Vuser** > **Parameter List** and select the desired parameter. The dialog box shows a read-only view of the parameter values.

File Path:	NewParam.dat	Browse...		
Create Data File				
Schema	Set 1	Set 2	Set 3	
[-] Addr				
[ABC] name	John Doe	Tom Smith	Kim Jones	
[ABC] street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.	
[ABC] city	Delray Beach			
[ABC] state	FL	AZ	MA	
[ABC] zipcode	33452		02134	
[-] phonenumber				
[+] birthday				

Edit Data

2 Open the Data Parameterization box.

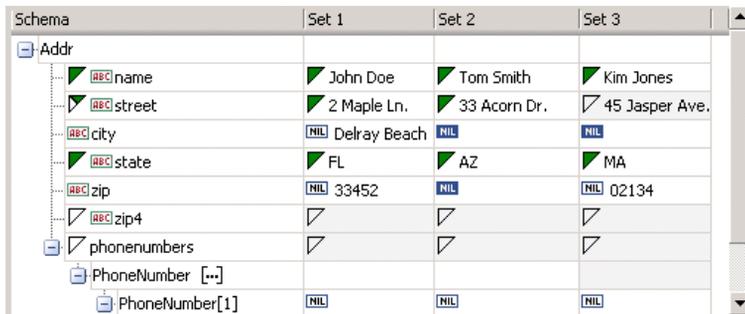
Click the **Edit Data** button to open the Data Parameterization dialog box.



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.

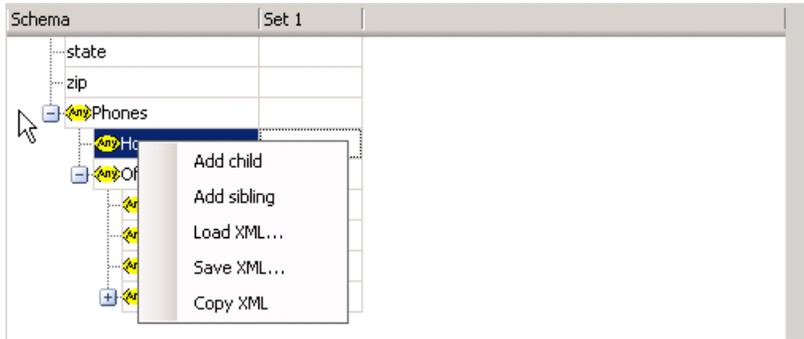
Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [..]			
PhoneNumber[1]	◆	◆	◆
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	◆	[]	◆
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneNumber[3]	[]	[]	[]
description	Mobile	Mobile	Mobile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

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 265.

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 "Data Assignment Methods for File/Table/XML Parameters" on page 263.

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 265.

4 If you chose **Unique** as the data assignment method the **When out of values** and **Allocate Vuser values in the Controller** options become enabled.

- **When out of values.** Specify what to do when there is no more unique data: **Abort Vuser**, **Continue in a cyclic manner**, or **Continue with last value**.
- **Allocate Vuser values in the Controller** (for LoadRunner users only). Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate x values for each Vuser**. For the second option, specify the number of values to allocate.

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

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

The list of input arguments is replaced by the parameter name, and ABC button is replaced 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 to 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 desired.

Tasks

How to Create a Parameter

This task describes how to create a parameter.

This task includes the following steps:

- "Select the argument that you want to parameterize" on page 277
- "Complete the Select or Create Parameter dialog box" on page 278
- "Add a list of desired values" on page 278
- "Modify the parameter braces - optional" on page 278

1 Select the argument that you want to parameterize

You can perform this step from both the script and tree views.

➤ Script view

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

- 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>Belement<C>Celement</C>`, select the whole string, `Belement<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.

➤ Tree view:

Select the step you want to parameterize, right-click and select **Properties** from the menu. The appropriate Step Properties dialog box opens.



Click the **ABC** icon next to the argument that you want to parameterize.

2 Complete the Select or Create Parameter dialog box

Specify the parameters name and type in the Select or Create Parameter dialog box. For user interface details, see "Select or Create Parameter Dialog Box" on page 282.

3 Add a list of desired values

From the Select or Create Parameter dialog box, select Properties. Create a table and add entries to serve as the list of values for your parameter. For user interface details, see "Parameter Properties Dialog Box" on page 284.

4 Modify the parameter braces - optional

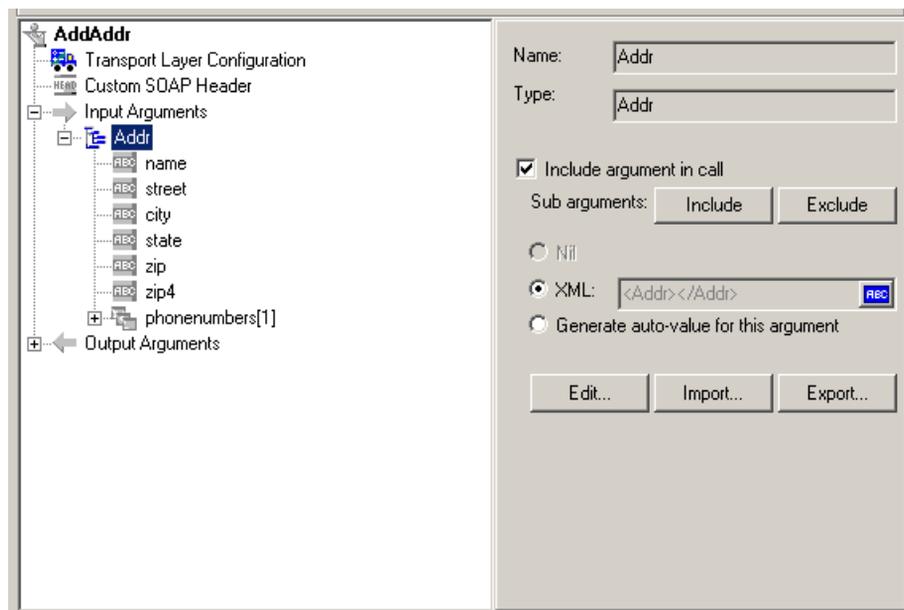
You can modify the braces that surround parameters by selecting **Tools** menu > **General Options** > **Parameterization** tab. For user interface details, see "Parameter Tab" on page 97.

How to Create an XML Parameter from a Web Service Call

This task describes how to create a new XML Parameter from a Web Service Call. This procedure is in addition to the standard procedure to create a parameter. XML Parameters can also be created by using the standard procedure.

To Create an XML Parameter from a Web Service Call:

- 1 Select the root element of the complex data structure. The right pane displays the argument's details.



- 2 Select **XML** in the right pane, and click the **ABC** icon. The Select or Create Parameter dialog box opens.
- 3 In the **Parameter name** box, enter a name for the parameter.
- 4 In the **Parameter type** box, select **XML** if it is not already selected.
- 5 Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

How to Work with Existing Parameters

This task describes how to replace strings with pre-existing parameters.

This task includes the following steps:

- "Replace a single string with a parameter" on page 280
- "Replace multiple strings with a parameter" on page 280

Replace a single string with a parameter

You can replace a single string with a pre-existing parameter by selecting the desired string while in script view and right-clicking and selecting **Use Existing Parameter**. Select a parameter or select **Select from Parameter List**.

Replace multiple strings with a parameter

You can replace a multiple occurrences of a string with a pre-existing parameter. To do this, replace at least one occurrence of the parameter and select it while in script view. Right-click the parameter and select **Replace more occurrences**. Use the Search and Replace dialog box to replace the desired occurrences of the string.

Restore the original string

You can undo a parameter and restore the original string by right-clicking the parameter in script view and selecting **Restore original value**.

How to Import Parameter Data from a Database

The following steps describe the ways in which you can import parameter data from an existing database. After you import the data, it is saved as a file with a .dat extension and stored as a regular parameter file.

- "Create a query using Microsoft Query" on page 281
- "Specify an SQL statement manually" on page 281

Create a query using Microsoft 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.

Specify an SQL statement manually

- 1 Select **Specify SQL Statement Manually** and click **Next**.
- 2 Click **Create** to specify a new connection string. The Select Data Source window opens.
- 3 Select a data source, or click **New** to create a new one. The wizard guides you through the procedure for creating an ODBC data source. When you are finished, the connection string appears in the **Connection String** box.
- 4 In the **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.

Reference

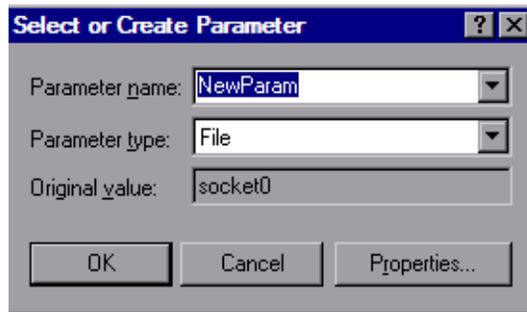
Parameter User Interface

This section includes (in alphabetical order):

- ▶ Select or Create Parameter Dialog Box on page 282
- ▶ Parameter Properties Dialog Box on page 284
- ▶ Parameter Simulation Dialog Box on page 297
- ▶ Parameter List Dialog Box on page 301
- ▶ Database Query Wizard on page 303

Select or Create Parameter Dialog Box

This dialog box enables you to create a new parameter or modify an existing parameter.



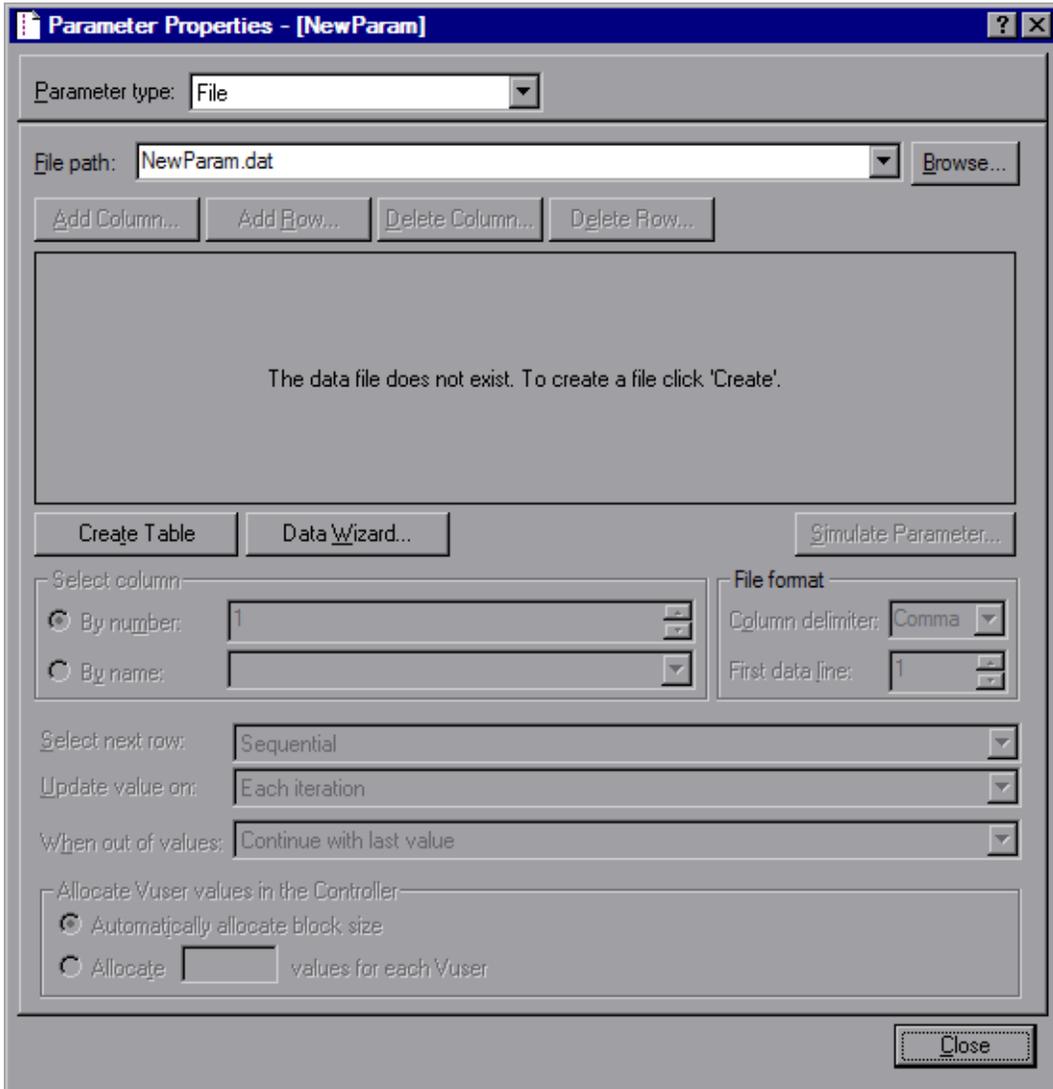
To access	Insert > New Parameter
Relevant tasks	"How to Create a Parameter" on page 277

User interface elements are described below:

UI Elements (A-Z)	Description
Parameter name	The name of the parameter. Note: Do not use the name unique , it is used by VuGen.
Parameter type	The type of the parameter. For information about the different parameter types see "Parameter Types" on page 260.
Original value	The original value of the parameter before parameterization.
	Opens the Parameter Properties dialog box. For details, see "Parameter Properties Dialog Box" on page 284.

Parameter Properties Dialog Box

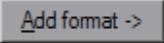
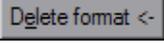
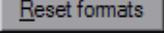
This page allows you to view and modify the properties of a parameter. This dialog box varies depending on the type of parameter you are using.



The screenshot shows the 'Parameter Properties - [NewParam]' dialog box. The 'Parameter type' is set to 'File'. The 'File path' is 'NewParam.dat', with a 'Browse...' button. Below the path are buttons for 'Add Column...', 'Add Row...', 'Delete Column...', and 'Delete Row...'. A large text area contains the message: 'The data file does not exist. To create a file click 'Create''. Below this are buttons for 'Create Table', 'Data Wizard...', and 'Simulate Parameter...'. The 'Select column' section has 'By number' selected with a value of '1', and 'By name' is unselected. The 'File format' section has 'Column delimiter' set to 'Comma' and 'First data line' set to '1'. The 'Select next row' is 'Sequential', 'Update value on' is 'Each iteration', and 'When out of values' is 'Continue with last value'. The 'Allocate User values in the Controller' section has 'Automatically allocate block size' selected, and 'Allocate' values for each User is set to an empty text box. A 'Close' button is at the bottom right.

To access	Right-click parameter > Parameter properties
-----------	--

Date/Time, Group Name, Iteration Number, Load Generation Name, and Vuser ID parameters

UI Elements (A-Z)	Description
	Adds the custom format specified in the Date/time format or Text format field to the format list.
	Deletes the selected format from the format list.
	Restores the format list to its default state.
Date/time format / Text format	You can specify a custom format here. See the chart below for a list of Date/time symbols.
Format list	The list of formats. See the chart below for a list of Date/time symbols.
Offset (Date/time type only)	Allows you to set an offset for the date/time parameter. For example, if you want to test a date next month, you set the date offset to 30 days. <ul style="list-style-type: none"> ▶ Working days only. Use values for work days only (excludes Saturdays and Sundays). ▶ Prior to current date. Sets the offset for a date or time that has already passed (negative offset).
Parameter type	The parameter type. For more information see "Parameter Types" on page 260.

UI Elements (A-Z)	Description
Sample (current time)	Displays an example parameter value based on the selected format.
Update values on	<ul style="list-style-type: none"> ▶ Each occurrence. Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. ▶ Each iteration. Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. ▶ Once. Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

The following table describes the date/time symbols:

Symbol	Description
c	complete date and time in digits
#c	complete date as a string and time
H	hours (24 hour clock)
I	hours (12 hour clock)
M	minutes
S	seconds

Symbol	Description
p	AM or PM
d	day
m	month in digits (01-12)
b	month as a string - short format (e.g. Dec)
B	month as a string - long format (e.g. December)
y	year in short format (e.g. 03)
Y	year in long format (e.g. 2003)

File parameters

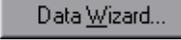
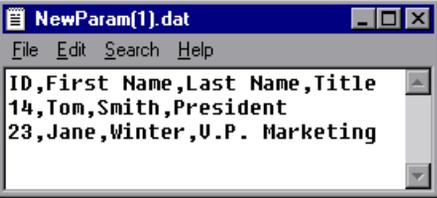
UI Elements (A-Z)	Description
A dd Column...	Adds a column to the data set.
A dd Row...	Adds a row to the data set.
C reate Table	Creates a new data table.
D ata Wizard...	Opens the Database Query Wizard, enabling you to import data from an existing database. For more information, see
D elete Column...	Deletes a column from the data set.
D elete Row...	Deletes a row from the data set.

UI Elements (A-Z)	Description
<p>Edit with Notepad...</p>	<p>Enables you to view and edit parameter values in Notepad. This is important when working with large data sets because VuGen only displays up to 100 rows in the UI.</p> <p>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).</p> 
<p>Simulate Parameter...</p>	<p>Opens the Parameter Simulation dialog box. This allows you to simulate the parameter behavior with your data set. For more information, see "Parameter Simulation Dialog Box" on page 297.</p>
<p>Select Column</p>	<p>Enables you to select the column to use as the data source either by the column number or name.</p>
<p>File Format</p>	<ul style="list-style-type: none"> ▶ Column delimiter. The character used to separate values in the data file. ▶ First data line. The first line of data to be used during Vuser script execution. The header is line 0. To begin with the first line after the header, specify 1. If there is no header, specify 0.
<p>Select next row</p>	<p>The method of selecting the file data during Vuser script execution. The options are: Sequential, Random, or Unique. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 263.</p>

UI Elements (A-Z)	Description
Update value on	The method that determines when the parameter will switch to the next value. The choices are Each Iteration , Each Occurrence , and Once . For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 263.
When out of values	Specify what to do when there is no more unique data: Abort the Vuser , Continue in a cyclic manner , or Continue with last value .
Allocate Vuser values in the Controller	(LoadRunner only). Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select Automatically allocate block size or Allocate x values for each Vuser . For the second option, specify the number of values to allocate. To track this occurrence, enable the Extended Log > Parameter Substitution option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <table_name>".
File path	Select the .dat file with the data for your parameter. Alternatively, you can create a new data set using the Create Table button.

Table parameters

UI Elements (A-Z)	Description
	Adds a column to the data set.
	Adds a row to the data set.
	Creates a new data table.

UI Elements (A-Z)	Description
	<p>Opens the Database Query Wizard, enabling you to import data from an existing database. For more information, see</p>
	<p>Deletes a column from the data set.</p>
	<p>Deletes a row from the data set.</p>
	<p>Enables you to view and edit parameter values in Notepad. This is important when working with large data sets because VuGen only displays up to 100 rows in the UI.</p> <p>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).</p> 
<p>Allocate Vuser values in the Controller</p>	<p>(LoadRunner only). Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select Automatically allocate block size or Allocate x values for each Vuser. For the second option, specify the number of values to allocate.</p> <p>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>".</p>

UI Elements (A-Z)	Description
Column	<p>specify which columns you want to use. Alternatively, you can select Select all columns.</p> <p>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.</p> <p>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.</p>
File path	<p>Select the .dat file with the data for your parameter. Alternatively, you can create a new data set using the Create Table button.</p>
Row delimiter for log display	<p>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.</p>
Rows	<ul style="list-style-type: none"> ▶ Rows per iteration. How many rows to use per iteration. This only relevant when the Update value on field is set to Each iteration. If Update value on is set to Once, then the same rows will be used for all iterations. ▶ First line of data. The first line of data to be used during script execution. To begin with the first line after the header, enter 1. ▶ Table information. Displays information about the table, including how many rows of data are available.
Select next row	<p>The method of selecting the file data during Vuser script execution. The options are: Sequential, Random, or Unique. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 263.</p>

UI Elements (A-Z)	Description
Update value on	The method that determines when the parameter will switch to the next value. The choices are Each Iteration , Each Occurrence , and Once . For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 263.
When not enough rows	Specifies what VuGen does when there are not enough rows in the table for the iteration. Example: The table you want to fill has 3 rows, but your data only has two rows. Select Parameter will get less rows than required to fill in only two rows. Select Use behavior of "Select Next Row" to loop around and get the next row according the method specified in the Select next row box.
When out of values	Specify what to do when there is no more unique data: Abort the Vuser , Continue in a cyclic manner , or Continue with last value .

BPT parameters

The BPT parameter type is only available with an HP Service Test license. For details, contact HP Support.

Random Number parameters

UI Elements (A-Z)	Description
Number format	Specifies the minimum number of digits your parameter will have. Where %01d represents one digit, %02d represents two digits, and so on.
Random range	The minimum and maximum range for the random values.

UI Elements (A-Z)	Description
Sample value	Displays an example parameter value based on the selected format.
Update value on	<ul style="list-style-type: none"> ▶ Each occurrence. Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. ▶ Each iteration. Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. ▶ Once. Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

Unique Number parameters

Note: When scheduling a scenario in the Controller, the **When out of values** option only applies to the **Run for HH:MM:SS** option in the Schedule Builder's Duration tab. It is ignored for the **Run until completion** option.

UI Elements (A-Z)	Description
Number format	Specifies the minimum number of digits your parameter will have. Where %01d represents one digit, %02d represents two digits, and so on.
Number range	<ul style="list-style-type: none"> ➤ Start. The starting value. ➤ Block size per Vuser. The amount of unique numbers assigned to each Vuser. For example, if you specify a starting value of 1 and a block size of 100, the values be 1-100 can be used by the first Vuser, the values 101-200 can be used by the second Vuser, etc.
Sample value	Displays an example parameter value based on the selected format.

UI Elements (A-Z)	Description
Update value on	<ul style="list-style-type: none"> ➤ Each occurrence. Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. ➤ Each iteration. Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. <ul style="list-style-type: none"> Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. ➤ Once. Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.
When out of values	<p>Determines what to do when the range of values is reached for a Vuser. The range of values is determined by the start value and the block size.</p> <p>Abort Vuser. Terminates the Vuser script.</p> <p>Continue in a cyclical manner. Restart the unique numbers for this Vuser from the beginning of its assigned range. For example, if a Vuser had the range of 1-100 and it reached 100, it would start again at 1.</p> <p>Continue with last value. Use the last assigned value for this parameter for all subsequent occurrences of this parameter. For example, if a Vuser had the range of 1-100 and it reached 100, it would continue with the value of 100 until the end of the script.</p>

User Defined Function parameters

UI Elements (A-Z)	Description
Function Name	The name of the function. Use the name of the function as it appears in the DLL file.
Library Names	The location of the relevant library files.
Update value on	<ul style="list-style-type: none"> ▶ Each occurrence. Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. ▶ Each iteration. Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. <ul style="list-style-type: none"> Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. ▶ Once. Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

XML parameters

For information about Web Services XML parameters, see "XML Parameters" on page 268.

Parameter Simulation Dialog Box

This dialog box allows you to view a simulation of your the behavior of a file parameter.

To access	Parameter List > Select Parameter > Simulate Parameter
<p>Important information</p>	<ul style="list-style-type: none"> ▶ This feature is only relevant for file type parameters. ▶ Not all types of Parameter Substitution can be simulated. If you select Select next row: Same line as... or Update value on: Each occurrence, then the Parameter Simulation dialog box will not open. ▶ VuGen can simulate up to 256 iterations and 256 Vusers. ▶ Run Indefinitely is compliant with the Real-life schedule in the Scheduler of the Controller. ▶ If you select Select next row: Unique in the Parameter List dialog, then each Vuser is assigned a unique range of rows from which the Simulator will substitute values (for that Vuser). <p>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.</p> <p>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.</p>

User interface elements are described below:

UI Elements (A-Z)	Description
Vusers	The number of Vusers to run in the simulation.
Scenario Run Mode	<ul style="list-style-type: none"> ▶ Run until completion. Enter the number of iteration to run or select Take number of iteration from Run-Time settings. ▶ Run indefinitely. Simulates the run indefinitely option in the controller. VuGen only actually simulates the number of iterations you specify.
	Runs the parameter simulation. The values of each parameter substitution are displayed.

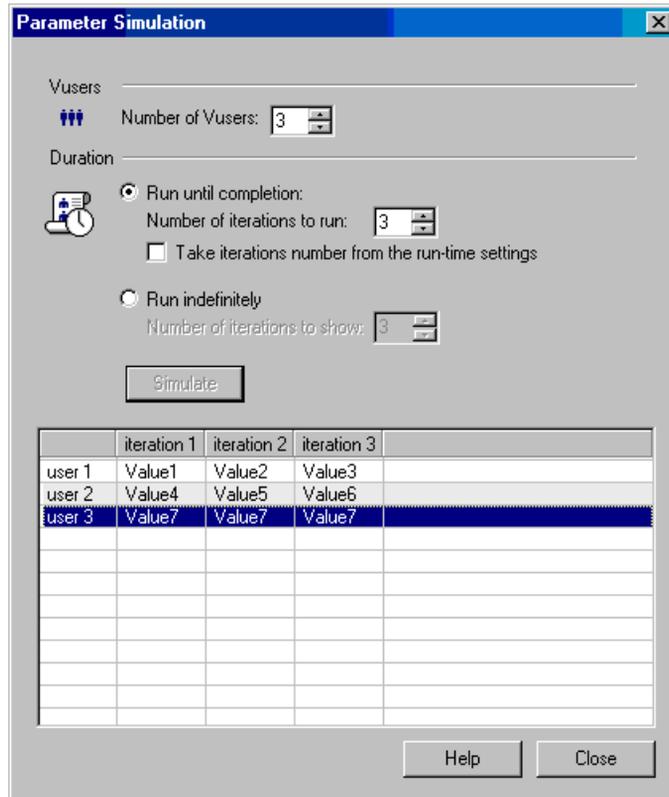
Example:

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

- ▶ **Values for the new parameter.** Value1 to Value7
- ▶ **Select next row.** Unique
- ▶ **When out of rows.** Continue with last value
- ▶ **Allocate Vuser values in the Controller.** Automatically allocate block size

Scenario run mode: Run until completion

In the following example, the user has selected three Vusers, set the Scenario run mode to **Run until completion**, and selected three iterations.



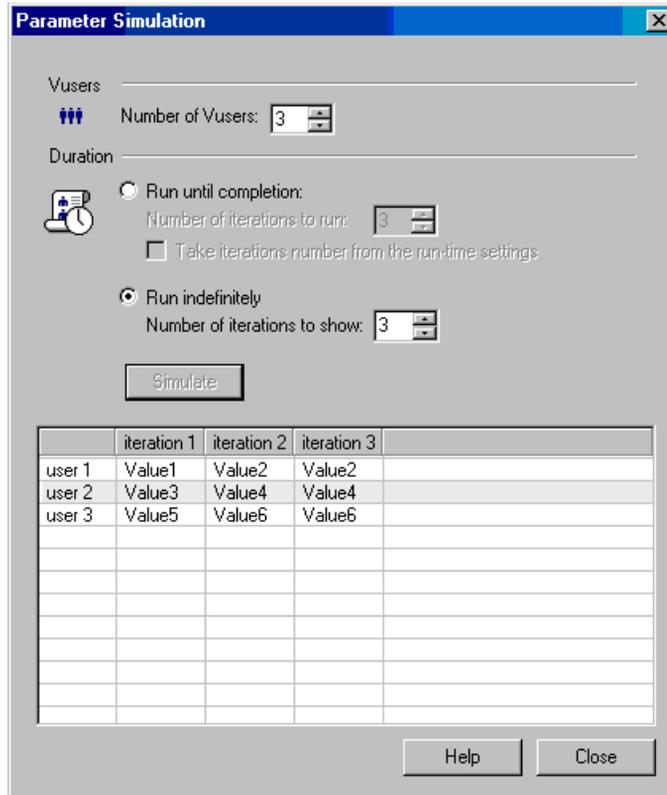
When the scenario run mode is set to **Run until completion**, the number of rows that each Vuser receives is the same as the number of iterations. The range allocation stops when there are no longer enough rows in the table.

As the simulation is run, the first Vuser takes the first three values (because this was the number of iterations). The second Vuser takes the next three values. The third Vuser takes the remaining value in the first iteration. For the remaining iterations, since the **When out of values** option in the Parameter List dialog box was set to **Continue with last value**, the third Vuser continues with the same value.

A fourth Vuser would have failed.

Scenario run mode: Run indefinitely

In the following example, the user has selected 3 Vusers and set the Scenario run mode to Run indefinitely and selected to show 3 iterations.



When the Scenario run mode is set to Run indefinitely, the allocated range for each Vuser is calculated by dividing the number of cells in the .dat file by the number of Vusers. In this scenario, that is $7/3 = 2$ (The simulator takes the closest smaller integer.).

As the simulation is run, the first Vuser takes Value1 and Value2. The second Vuser takes Value3 and Value4 and the third Vuser takes Value5 and Value6. Since there are were only 3 Vusers, Value7 was not distributed.

Note: If you hold the mouse over the cells in the first column of the table, a tool tip appears with information about which values were assigned to that Vuser.

If you hold the mouse over cells which were not assigned values, a tool tip appears with the reason no values were assigned.

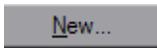
A tool tip does not appear if a proper value was assigned.

Parameter List Dialog Box

This dialog box enables you to view, create, delete, select, and modify parameters. The Parameter list shows all of the parameters that you created, including both input and output parameters.

To access	Vuser > Parameter List
Important information	Do not name a parameter unique, since this name is used by VuGen.

User interface elements are described below:

UI Elements (A-Z)	Description
	Creates a new parameter. This does not replace any highlighted text with the parameter. Do not name a parameter <i>unique</i> , since this name is used by VuGen.
	Deletes the selected parameter.

UI Elements (A-Z)	Description
<p><Parameter Properties Pane></p>	<p>This pane appears different depending on the type of parameter you are using. For information about this pane, see one of the following sections:</p> <ul style="list-style-type: none"> ▶ For Date/Time, Group Name, Iteration Number, Load Generation Name, and Vuser ID parameters see "Date/Time, Group Name, Iteration Number, Load Generation Name, and Vuser ID parameters" on page 285. ▶ For File parameters, see "File parameters" on page 287. ▶ For Table parameters, see "Table parameters" on page 289. ▶ For BPT parameters, see "BPT parameters" on page 292. ▶ For Random Number parameters, see "Random Number parameters" on page 292. ▶ For Unique Number parameters, see "Unique Number parameters" on page 294. ▶ For User Defined Function parameters, see "User Defined Function parameters" on page 296. ▶ For XML parameters, see "XML parameters" on page 297.
<p>Parameter type</p>	<p>This drop-down list lets you select the parameter type. For information about the different parameter types, see "Parameter Types" on page 260.</p>

Database Query Wizard

This wizard enables you to select data for a parameter form an existing database.

To access	Right-click parameter > Parameter properties > Data Wizard (only available for File / Table type parameters)
Relevant tasks	"How to Import Parameter Data from a Database" on page 280

User interface elements are described below:

UI Elements (A-Z)	Description
Query Definition	Select from one of the following options: <ul style="list-style-type: none"> ▶ Create query using Microsoft Query ▶ Specify SQL statement manually
Show me how to use Microsoft Query	Displays instructions for using Microsoft Query when after you click next.
Maximum number of rows	The maximum number of rows to be created in the .dat file based on the specified query.

Troubleshooting and Limitations

This section describes troubleshooting and limitations for parameters.

Function argument limitations

You can use parameterization only for the arguments within a function. You cannot parameterize text strings that are not function arguments. In addition, not all function arguments can be parameterized. For details on which arguments you can parameterize, see the *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*.

10

Recording Options

This chapter includes:

Concepts

- ▶ Port Mapping Overview on page 308
- ▶ Port Mapping Auto Detection on page 308
- ▶ EUC-Encoding (Japanese Windows only) on page 310
- ▶ Script Generation Preference Overview on page 311
- ▶ Script Language Options on page 312
- ▶ Recording Levels Overview on page 312
- ▶ Serialization Overview on page 316
- ▶ Tips for Working with Event Listening and Recording on page 316

Reference

- ▶ Example of Click and Script Out of Context Recording on page 318
- ▶ Protocol Compatibility Table on page 319
- ▶ Recording Options User Interface on page 323

Concepts

Port Mapping Overview

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 node will be available. The only exception is when you record HTTP or WinSock as a single protocol script.

Port Mapping Auto Detection

VuGen's advanced port-mapping options let you configure the **auto-detection** options. VuGen's auto-detection analyzes the data that is sent to the server. It checks the data for a signature, a pattern in the data's content, that identifies the protocol. For the purpose of detecting a signature, all of the send buffers until the first receive buffer, are combined. All send buffers that were sent until a receive buffer is returned, are considered a single data **transition**. By default, no mappings are defined and VuGen employs auto-detection. In some protocols, VuGen determines the

type in a single transition, (such as HTTP). Other network protocols require several transitions before determining the type. For this purpose, VuGen creates a temporary buffer for each server-port combination. If VuGen cannot determine the protocol type by reading the first transition buffers, it stores the data in a temporary buffer. It continues to read the incoming buffers until it detects a signature of a specific protocol.

By default, VuGen allows 4 transitions and uses a temporary buffer of 2048 bytes in order to detect a protocol signature. If VuGen has not yet determined the type after reaching the maximum number of transitions, or after reaching the maximum buffer size, it assigns the data to the WinSock protocol. If you did not instruct VuGen to record the WinSock protocol (in the multi-protocol selection), VuGen discards the data.

You can change the maximum number of buffers you want VuGen to read in order to detect the protocol type. You can also specify the size of the temporary buffer. In instances where the amount of data in the first send buffers, is greater than the size of the temporary buffer, VuGen cannot auto-detect the protocol type. In this case, you should increase the size of the temporary buffer.

When working with the above network level protocols, we recommend that you allow VuGen to use auto-detection to determine the protocol type. In most cases, VuGen's recorder is able to recognize the signatures of these protocols. It then automatically processes them according to the protocol specifications. In certain instances, however, VuGen may be unable to recognize the protocol. For example:

- ▶ The protocol signature closely resembles an existing protocol, resulting in erroneous processing.
- ▶ There is no unique signature for the protocol.
- ▶ The protocol uses SSL encryption, and therefore cannot be recognized on a WinSock level.

In all of the above cases, you can supply information to uniquely identify the server and port hosting the protocol.

EUC-Encoding (Japanese Windows only)

When working with non-Windows standard character sets, you may need to perform a code conversion. A character set is a mapping from a set of characters to a set of integers. This mapping forms a unique character-integer combination for a given alphabet. Extended UNIX Code (EUC) and Shift Japan Industry Standard (SJIS) are non-Windows standard character sets used to display Japanese characters on Web sites.

Windows uses SJIS encoding, while UNIX uses EUC encoding. When a Web server is running UNIX and the client is running Windows, the characters in a Web site are not displayed on the client machine properly due to the difference in the encoding methods. This affects the display of EUC-encoded Japanese characters in a Vuser script.

During recording, VuGen detects the encoding of a Web page through its HTTP header. If the information on the character set is not present in the HTTP header, it checks the HTML meta tag.

If you know in advance that a Web page is encoded in EUC, you can instruct VuGen to use the correct encoding by using the recording options. To record a page in EUC-encoding, enable the **EUC** option in the Recording Options **Recording** node (only visible for Japanese Windows).

Enabling the **EUC** option forces VuGen to record a Web page in EUC encoding, even when it is not EUC-encoded. Therefore, you should only enable this option when VuGen cannot detect the encoding from the HTTP header or the HTML meta tag or when you know in advance that the page is EUC-encoded.

During recording, VuGen receives an EUC-encoded string from the Web server and converts it to SJIS. The SJIS string is saved in the script's **Action** function. However, for replay to succeed, the string has to be converted back to EUC before being sent back to the Web server. Therefore, VuGen adds a **web_sjis_to_euc_param** function before the **Action** function, which converts the SJIS string back to EUC.

In the following example, the user navigates to an EUC-encoded Web page and clicks a link. VuGen records the **Action** function and adds the **web_sjis_to_euc_param** function to the script before the **Action** function.

```
web_sjis_to_euc_param("param_link","Search");  
web_link("LinkStep","Text={param_link}");
```

For more information, see "Advanced URL Dialog Box" on page 349.

Script Generation Preference Overview

Before you record a session, VuGen allows you to specify a language for script generation. The available languages for script generation vary per protocol. Some of the available languages are C, C#, Visual Basic, Visual Basic .NET, VB Script, and Javascript. By default, VuGen generates a script in the most common language for that protocol, but you can change this through the **Script** recording options node.

For user interface details, see "General Script Node" on page 353.

Tip: If you record a script in one language, you can regenerate it in another language after the recording. For task details, see "How to Regenerate a Vuser Script" on page 111.

After you select a generation language, you can enable language-specific recording options which instruct the recorder what to include in the script and how to generate it.

If at least one of the protocols you are recording has multi-protocol capabilities, the Script node will be available except when you record HTTP or WinSock as a single protocol script.

Script Language Options

When you record a session, VuGen creates a script that emulates your actions. The default script generation language is C or C# for MS .NET. For the FTP, COM/DCOM, and mail protocols (IMAP, POP3, and SMTP), VuGen can also generate a script in Visual Basic, VB Script, and Javascript. The following list specifies which protocols are appropriate for each language:

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

Recording Levels Overview

VuGen lets you specify what information to record and which functions to use when generating a Vuser script by selecting a recording level in the **General Recording Levels** node of the **Recording Options dialog box**. The recording level you select depends on your needs and environment. The available levels are **GUI-based script**, **HTML-based script**, and **URL-based script**. For user interface information, see "General Recording Node" on page 348.

The following examples show scripts using the three recording levels:

GUI-based script

Records HTML actions as context sensitive GUI functions, for example `web_text_link`.

```
/* GUI-based mode - CS type functions with JavaScript support*/
vuser_init()
{
web_browser("WebTours",
            DESCRIPTION,
            ACTION,
            "Navigate=http://localhost:1080/WebTours/",
            LAST);

web_edit_field("username",
              "Snapshot=t2.inf",
              DESCRIPTION,
              "Type=text",
              "Name=username",
              "FrameName=navbar",
              ACTION,
              "SetValue=jojo",
              LAST);
...
}
```

HTML-based script

Generates a separate step for each HTML user action. The steps are intuitive, but they do not reflect true emulation of the JavaScript code.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("WebTours",
        "URL=http://localhost/WebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
...
```

URL-based script

Records all browser requests and resources from the server that were sent due to the user's actions. Automatically records all HTTP resources as URL steps (**web_url** statements). For normal browser recordings, it is not recommended to use the URL-based mode since is more prone to correlation related issues. However, if you are recording pages such as applets and non-browser applications, this mode is ideal.

URL-based scripts are not as intuitive as the HTML-based scripts since all actions are recorded as **web_url steps** instead of **web_link**, **web_image**, and so on.

```
/* URL-based mode - only web_url functions */
...
web_url("spacer.gif",
        "URL=http://graphics.hplab.com/images/spacer.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=",
        "Mode=HTTP",
        LAST);

web_url("calendar_functions.js",
        "URL=http://www.im.hplab.com/travelp/calendar_functions.js",
        "Resource=1",
        "RecContentType=application/x-javascript",
        "Referer=",
        "Mode=HTTP",
        LAST);
...
```

You can switch recording levels and advanced recording options while recording, provided that you are not recording a multi-protocol script. The option of combining recording levels is available to advanced users for performance testing.

You can also regenerate a script after recording, using a different method than the original recording. For example, if you 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.

Serialization Overview

VuGen uses serialization when it encounters an unknown object during the recording, provided that the object supports serialization. An unknown object can be an input argument which was not included by the filter and therefore its construction was not recorded. Serialization helps prevent compilation errors caused by the passing of an unknown argument to a method. If an object is serialized, it is often advisable to set a custom filter to record this object.

Tips for Working with Event Listening and Recording

It can sometimes be difficult to find the ideal listen and recording settings. When defining these settings, keep in mind the following guidelines:

- ▶ To record an event on an object, you must instruct VuGen to listen for the event, and to record the event when it occurs. You can listen for an event on a child object, even if a parent object contains the handler or behavior, or you can listen for an event on a parent object, even if the child object contains the handler or behavior.

However, you must enable recording for the event on the source object (the one on which the event actually occurs, regardless of which parent object contains the handler or behavior).

For example, suppose a table cell with an **onmouseover** event handler contains two images. When a user touches either of the images with the mouse pointer, the event bubbles up to the cell and includes information on which image was actually touched. You can record this mouseover event by:

- ▶ Setting **Listen** on the WebTable mouseover event to **If Handler** (so that VuGen "hears" the event when it occurs), while disabling recording on it, and then setting **Listen** on the Image mouseover event to **Never**, while setting its recording status to **Enable** (to record the mouseover event on the image after it is listened to at the WebTable level).

- ▶ Setting **Listen** on the Image mouseover event to **Always** (to listen for the mouseover event even though the image tag does not contain a behavior or handler), and setting the recording status on the Image object to **Enabled** (to record the mouseover event on the image).
- ▶ Instructing VuGen to listen for many events on many objects may lower performance, so try to limit listening settings to the required objects.
- ▶ In rare situations, listening to the object on which the event occurs (the source object) may interfere with the event.

Reference

Example of Click and Script Out of Context Recording

In the following example, a script was regenerated with the out-of-context recording option enabled.

```
web_image_link("Search Flights Button",
    "Snapshot=t5.inf",
    DESCRIPTION,
    "Alt=Search Flights Button",
    "FrameName=navbar",
    ACTION,
    "ClickCoordinates=58,9",
    LAST);

web_add_cookie("MSO=SID&1141052844; DOMAIN=localhost");

web_add_cookie("MTUserInfo=hash&47&firstName&Joseph&expDate&%0A&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);
```

For more information, see "GUI Properties Advanced Node" on page 356

Protocol Compatibility Table

The following table lists the types of Vuser scripts and which recording options nodes are available for each type.

Protocol	Recording Options Nodes
AMF	<ul style="list-style-type: none"> ▶ General - Script, Protocols, Recording ▶ AMF - Code Generation ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
AJAX	<ul style="list-style-type: none"> ▶ General - Script, Recording ▶ GUI Properties - Advanced, Web Event Configuration ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
C Vuser	<ul style="list-style-type: none"> ▶ None
Citrix	<ul style="list-style-type: none"> ▶ General - Script ▶ Citrix - Configuration, Recorder, Code Generation, Login

Protocol	Recording Options Nodes
COM/DCOM	<ul style="list-style-type: none"> ▶ General - Script ▶ COM/DCOM - Filter, Options
DB2 CLI	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
DNS	<ul style="list-style-type: none"> ▶ None
EJB	<ul style="list-style-type: none"> ▶ Java Environment Settings - Classpath ▶ EJB Options - Code Generation Options
FTP	<ul style="list-style-type: none"> ▶ General - Script ▶ Network - Port Mapping
Flex	<ul style="list-style-type: none"> ▶ General - Script, Protocols, Recording ▶ Flex - Code Generation ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
i-mode	<ul style="list-style-type: none"> ▶ General - Recording ▶ HTTP Properties - i-mode Toolkit, Advanced, Correlation
Informix	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
IMAP	<ul style="list-style-type: none"> ▶ General - Script ▶ Network - Port Mapping
Java over HTTP	<ul style="list-style-type: none"> ▶ General - Recording ▶ Java Environment Settings - Java VM, Classpath ▶ Traffic Analysis - Traffic Filters ▶ Network - Port Mapping ▶ HTTP Properties - Browser, Recording Proxy, Advanced, Correlation
Java Record Replay	<ul style="list-style-type: none"> ▶ Java Environment Settings - Java VM, Classpath ▶ Recording Properties - Recorder Options, Serialization Options, Correlation Options, Log Options, Corba Options
Javascript Vuser	<ul style="list-style-type: none"> ▶ None

Protocol	Recording Options Nodes
LDAP	<ul style="list-style-type: none"> ▶ General - Script
MMS (Media Player)	<ul style="list-style-type: none"> ▶ None
.NET	<ul style="list-style-type: none"> ▶ General - Script ▶ .NET - Recording, Filters
RDP	<ul style="list-style-type: none"> ▶ General - Script ▶ RDP - Login, Code Generation (Basic, Advanced, and Agent) ▶ Network - Port Mapping
MAPI	<ul style="list-style-type: none"> ▶ None
MS SQL Server	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
MMS (Multimedia Messaging Service)	<ul style="list-style-type: none"> ▶ None
ODBC	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
Oracle (2-Tier)	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
Oracle NCA	<ul style="list-style-type: none"> ▶ General - Script, Protocols, Recording ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
Oracle Web Applications 11i	<ul style="list-style-type: none"> ▶ General - Script, Recording ▶ GUI Properties - Advanced, Web Event Configuration ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
PeopleSoft Enterprise	<ul style="list-style-type: none"> ▶ General - Script, Recording ▶ GUI Properties - Advanced, Web Event Configuration ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation

Protocol	Recording Options Nodes
PeopleSoft Tuxedo	<ul style="list-style-type: none"> ▶ None
POP3	<ul style="list-style-type: none"> ▶ General - Script ▶ Network - Port Mapping
Real	<ul style="list-style-type: none"> ▶ General - Script
SAP Web	<ul style="list-style-type: none"> ▶ General - Script, Recording ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
SAP Click and Script	<ul style="list-style-type: none"> ▶ General - Script, Recording ▶ GUI Properties - Advanced, Web Event Configuration ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
SAPGUI	<ul style="list-style-type: none"> ▶ General - Script ▶ SAPGUI - General, Code Generation, Auto Logon
Siebel Web	<ul style="list-style-type: none"> ▶ General - Script, Protocols, Recording ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
Silverlight	<ul style="list-style-type: none"> ▶ General - Script, Protocols, Recording ▶ Data Format Extensions - Configuration, Headers Chain, Body Chain, Cookies Chain, Query String Chain ▶ Silverlight - Services ▶ Network - Port Mapping ▶ HTTP Properties - Advanced, Correlation
SMTP	<ul style="list-style-type: none"> ▶ General - Script ▶ Network - Port Mapping
Sybase CTlib / DBlib	<ul style="list-style-type: none"> ▶ General - Script ▶ Database - Database
RTE	<ul style="list-style-type: none"> ▶ RTE - Configuration, RTE
Tuxedo, Tuxedo 6	<ul style="list-style-type: none"> ▶ None

Protocol	Recording Options Nodes
VB Script Vuser	➤ None
VB Vuser	➤ None
WAP	<ul style="list-style-type: none"> ➤ General - Script, Protocols, Recording ➤ Network - Port Mapping ➤ HTTP Properties - Advanced, Correlation
Web (Click and Script)	<ul style="list-style-type: none"> ➤ General - Script, Recording ➤ GUI Properties - Advanced, Web Event Configuration ➤ Network - Port Mapping ➤ HTTP Properties - Advanced, Correlation
Web (HTTP/HTML)	<ul style="list-style-type: none"> ➤ General - Script, Protocols, Recording ➤ Network - Port Mapping ➤ HTTP Properties - Advanced, Correlation ➤ Data Format Extensions - Configuration, Headers Chain, Body Chain, Cookies Chain, Query String Chain
Web Services	<ul style="list-style-type: none"> ➤ General - Script, Protocols, Recording ➤ Traffic Analysis - Traffic Filters ➤ Network - Port Mapping ➤ HTTP Properties - Advanced, Correlation
Windows Sockets	➤ Sockets - Winsock

Recording Options User Interface

This section includes:

- AMF Code Generation Node on page 325
- Citrix Code Generation Node on page 327
- Citrix Configuration Node on page 328
- Citrix Login Node on page 328
- Citrix Recorder Node on page 331

- ▶ COM/DCOM Filter Node on page 333
- ▶ COM/DCOM Options Node on page 339
- ▶ Database Node on page 340
- ▶ Data Format Extensions Configurations Node on page 343
- ▶ Data Format Extensions - Chain Nodes on page 344
- ▶ EJB Code Generation Options Node on page 346
- ▶ Flex Code Generation Node on page 346
- ▶ General Protocol Node on page 348
- ▶ General Recording Node on page 348
- ▶ General Script Node on page 353
- ▶ GUI Properties Advanced Node on page 356
- ▶ GUI Properties Web Event Configuration Node on page 358
- ▶ HTTP Advanced Node on page 362
- ▶ HTTP Correlation Node on page 368
- ▶ Java Classpath Node on page 373
- ▶ Java VM Node on page 374
- ▶ Microsoft .NET Filters Node on page 375
- ▶ Filter Manager on page 377
- ▶ Add Reference Dialog Box on page 381
- ▶ Microsoft .NET Recording Node on page 382
- ▶ Network Port Mapping Node on page 388
- ▶ RDP Code Generation - Advanced Node on page 393
- ▶ RDP Code Generation - Agent Node on page 394
- ▶ RDP Code Generation - Basic Node on page 395
- ▶ RDP Login Node on page 397
- ▶ Recording Properties Corba Options Node on page 398
- ▶ Recording Properties Correlation Options Node on page 399

- Recording Properties Log Options Node on page 400
- Recording Properties Recorder Options Node on page 401
- Recording Properties Serialization Options Node on page 403
- RTE Configuration Node on page 405
- RTE Node on page 405
- SAPGUI Auto Logon Node on page 407
- SAPGUI Code Generation Node on page 407
- SAPGUI General Node on page 408
- Silverlight Services Node on page 409
- WinSocket Node on page 413

AMF Code Generation Node

Enables you to set the code generation settings for the AMF protocol.

To Access	Tools > Recording Options > AMF > Code Generation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Encode externalizable object with LoadRunner parser	<p>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.</p> <p>To overcome this issue, VuGen generates Custom Request functions containing unparsed AMF3 binary data when it encounters an externalizable object.</p> <p>You can attempt to encode these objects using the LoadRunner parser as well. This decodes all externalizable objects as standard AMF3 objects, generating amf_call functions. However, if objects cannot be parsed, they are not decoded at all so you have to check the script after it is generated to make sure that the parser was effective. Additionally, this option may reduce the stability of code generation.</p> <p>Default value: enabled.</p>

Citrix Code Generation Node

Enables you to configure the way VuGen captures information during recording.

To Access	Tools > Recording Options > Citrix > Code Generation
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ▶ Text synchronization steps that you add manually during the recording are not affected by the above settings—they appear in the script even if you disable the above options. ▶ The synchronization options also work for regenerating a script. For example, if you originally recorded a script with Add text synchronization calls disabled, you can regenerate after to recording to include text synchronization.

User interface elements are described below:

UI Element (A-Z)	Description
Use Citrix Agent input in Code Generation	<p>Use the Citrix Agent input to generate a more descriptive script with additional synchronization functions.</p> <p>Default value: enabled.</p> <ul style="list-style-type: none"> ▶ Add text synchronization calls. Adds text synchronization Sync on Text steps before each mouse click. <p>Default value: disabled.</p>

Citrix Configuration Node

Enables you to set the window properties and encryption settings for the Citrix client during the recording session.

To Access	Tools > Recording Options > Citrix > Configuration
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Encryption Level	The level of encryption for the ICA connection: Basic, 128 bit for login only, 40 bit, 56 bit, 128 bit, or Use Server Default to use the machine's default.
Window Size	The size of the client window. Default value: 800 x 600.

Citrix Login Node

Enables you to you set the connection and login information for the recording session.

To Access	Tools > Recording Options > Citrix > Login
Important Information	<ul style="list-style-type: none"> ➤ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ➤ If you do not provide login information, you are prompted for the information when the client locates the specified server.

User interface elements are described below:

UI Element (A-Z)	Description
<p>Connection</p>	<ul style="list-style-type: none"> ▶ Network Protocol. The preferred protocols are TCP/IP and TCP/IP+HTTP. Most Citrix Servers support TCP/IP, however Citrix Clients starting with 11.2 do not. Certain servers, however, are configured by the administrators to allow only TCP/IP with specific HTTP headers. If you encounter a communication problem, select the TCP/IP+HTTP option. ▶ Server. The Citrix server name. To add a new server to the list, click Add, and enter the server name (and its port for TCP/IP + HTTP). <p>Note: Multiple servers apply only when you specify a Published Application. If you are connecting to the desktop without a specific application, then list only one server.</p> ▶ Published Application. The name of the Published Application as it is recognized on Citrix server. The drop-down menu contains a list of the available applications. If you do not specify a published application, VuGen uses the server's desktop. If you added or renamed a published application, close the Recording options and reopen them to view the new list. Additionally, you can also enter the name of a published application manually if you know it exists (useful in cases where the drop-down list is inaccurate). <p>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.</p> <p>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.</p>
<p>Define connection parameters</p>	<p>Allows you to manually define the logon and connection information.</p>

UI Element (A-Z)	Description
Logon Information	Specify the User Name , Password , and Domain of the citrix user. Optionally, you can also specify the Client Name by which the MetaFrame server identifies the client.
Use ICA file for connection parameters	Specify an ICA file with the log configuration information.

Citrix Recorder Node

Enables you to specify how to generate window names where the window titles change during recording. You can also specify whether to save snapshots of the screens together with the script files and whether to generate text synchronization functions.

To Access	Tools > Recording Options > Citrix > Recorder
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Save snapshots	Saves a snapshot of the Citrix client window for each script step, when relevant. We recommend that you enable this option to provide you with a better understanding of the recorded actions. Saving snapshots, however, uses more disk space and slows down the recording session.
Window name	<p>In some applications, the active window name changes while you are recording. If you try to replay the script as is, the Vuser uses the original window name and the replay may fail. You can specify a naming convention for the windows in which VuGen uses a common prefix or common suffix to identify the windows as follows:</p> <ul style="list-style-type: none"> ➤ 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. <p>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.</p> <p>Example: <code>ctrx_sync_on_window ("My Application*", ACTIVATE, ...CTRX_LAST);</code></p>

 **COM/DCOM Filter Node**

This dialog box enables you to set the

To access	Tools > Recording Options > COM/DCOM > Filter
-----------	---

User interface elements are described below:

UI Elements (A-Z)	Description
DCOM Profile	<p>Specify one of the following filter types:</p> <ul style="list-style-type: none">▶ 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. <p>You can also save the current settings and delete a filter using the Save As and Delete buttons.</p>

UI Elements (A-Z)	Description
DCOM Listener Settings List	<p data-bbox="582 222 1215 348">Displays a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.</p> <p data-bbox="582 366 1215 517">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.</p> <p data-bbox="582 534 1215 725">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.</p> <p data-bbox="582 743 1215 829">Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.</p> <ul style="list-style-type: none"> <li data-bbox="582 847 1215 939">➤ Environment. The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record. <li data-bbox="582 947 1215 1098">➤ 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. <p data-bbox="582 1116 1215 1203">Type Libraries. In the lower section of the dialog box, VuGen displays the following information for each type library.</p> <ul style="list-style-type: none"> <li data-bbox="611 1220 1215 1255">➤ TypLib. The name of the type library (tlb file). <li data-bbox="611 1263 1215 1298">➤ Path. The path of the type library. <li data-bbox="611 1307 1215 1359">➤ Guid. The Global Unique Identifier of the type library.

UI Elements (A-Z)	Description
	<p>Adds another COM type library.</p> <ul style="list-style-type: none"> ▶ Browse Registry. Displays a list of type libraries found in the registry of the local computer. Select the check box next to the desired library or libraries and click OK. ▶ Browse file system. Allows you to select type libraries from your local file system. ▶ Browse MTS. add a component from a Microsoft Transaction Server. The MTS Components dialog box prompts you to enter the name of the MTS server. Type the name of the MTS server and click Connect. Remember that to record MTS components you need an MTS client installed on your machine. <p>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.</p>

UI Elements (A-Z)	Description
	Removes a COM type library.
	<p>Excludes interfaces in the filter through the Excluded Interfaces dialog box.</p> <p>In this dialog box, the checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click Add Interface... in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the Add Interface... feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter.</p> <p>An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the a warning. If you check Don't ask me again and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click Yes to all to change the status of all instances of this interface for all other classes, click No to all to leave the status of all other instances unchanged. Click Next Instance to view the next class that uses this interface.</p>

COM/DCOM Options Node

This dialog box enables you to set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.

To access	Tools > Recording Options > COM/DCOM > Options
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
ADO Recordset filtering	Condense multiple recordset operations into a single-line fetch statement (enabled by default).
Declare Temporary VARIANTS as Globals	Define temporary VARIANT types as Globals, not as local variables (enabled by default).
Fill array in separate scopes	Fill in each array in a separate scope (enabled by default).
Fill structure in separate scopes	Fill in each structure in a separate scope (enabled by default).
Generate COM exceptions	Generate COM functions and methods that raised exceptions during recording (disabled by default).
Generate COM statistics	Generate recording time performance statistics and summary information (disabled by default).
Limit size of SafeArray log	Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).
Release COM Objects	Record the releasing of COM objects when they are no longer in use (enabled by default).
Save Recordset content	Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).
Trap binded moniker objects	Trap all of the bound moniker objects (disabled by default).

Database Node

Enables you to set the recording options for database protocols.

To Access	Tools > Recording Options > Database > Database
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Advanced Recording Options Dialog Box.
Automatic transactions	Marks every lrd_exec and lrd_fetch function as a transaction. When these options are enabled, VuGen inserts lr_start_transaction and lr_end_transaction functions around every lrd_exec or lrd_fetch function. Default value: Disabled.
Script options	Generates comments into recorded scripts, describing the lrd_stmt option values. In addition, you can specify the maximum length of a line in the script. Default value: 80 characters.
Think time	VuGen automatically records the operator's think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an lr_think_time statement before LRD functions. If the recorded think time is below the threshold level, an lr_think_time statement is not generated. Default value: five seconds.

 **Advanced Recording Options Dialog Box**

Enables you set the advanced recording options for database protocols.

To Access	Tools > Recording Options > Database > Database > Advanced
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Code generation buffer size	<p>Specify in kilobytes the maximum size of the code generation buffer.</p> <p>Default value: 128 kilobytes.</p>
CtLib Function	<p>You can instruct VuGen to generate a send data time stamp or an extended result set statement.</p> <ul style="list-style-type: none"> ▶ Generate send data time stamp. Generates lrd_send_data statements with the TotalLen and Log keywords for the mpszReqSpec parameter. The Advanced Recording Options dialog box lets you instruct VuGen to also generate the TimeStamp keyword. If you change this setting on an existing script, you must regenerate the Vuser script by choosing 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 be correlated with a timestamp generated by an earlier lrd_send_data. ▶ Generate extended result set statement. Generates an lrd_result_set function when preparing the result set. This setting instructs VuGen to generate the extended form of the lrd_result_set function, lrd_result_set_ext. In addition to preparing a result set, this function also issues a return code and type from ct_results.

UI Element (A-Z)	Description
Recording engine	<p>You can instruct VuGen to record scripts with the older LRD recording engine for compatibility with previous versions of VuGen.</p> <p>Note: This option is only available for single-protocol scripts.</p>
Recording log options	<p>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.</p> <p>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.</p>

Data Format Extensions Configurations Node

Enables you to configure some general settings relating to Data Format Extensions.

To Access	Tools > Recording Options > Data Format Extensions > Configurations
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
See Also	"Data Format Extensions" on page 893 "Data Format Extension List" on page 901

User interface elements are described below:

UI Element	Description
Enable Data Format Extension	Enables Data Format Extensions during code generation. For more information, see "Data Format Extensions" on page 893.
Code Generation	<ul style="list-style-type: none"> ▶ Format code and snapshots. Enables Data Format Extensions on the code and snapshot data. ▶ Format snapshots only. Enables Data Format Extensions on snapshot data, but does not format the data in the script itself.
Verify Formatted Data	<p>Checks the results of the formatted data by converting it back to the original state and verifying that it matches the original data.</p> <p>Note: This option is not available for all extensions. Currently it is available for the Base64 extension.</p>

Data Format Extensions - Chain Nodes

Enables you to add, remove, and modify the Data Format Extensions on the Headers, Body, Cookies, and Query String chains.

To Access	Tools > Recording Options > Data Format Extensions > Configurations
Important Information	<p>These nodes are identical, but they represent different chains. For information about each chain, see the table below.</p> <p>These nodes are only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.</p>
See Also	<p>"Data Format Extensions" on page 893</p> <p>"Data Format Extension List" on page 901</p>

User interface elements are described below:

UI Element (A-Z)	Description
	Add a Data Format Extension to the extension chain.
	Remove a Data Format Extension from the extension chain.
	Move a Data Format Extension up or down in the extension chain. Extensions are run in the order in which they appear in this list.
<Data Format Extension list>	<p>An ordered list of Data Format Extensions. VuGen will use these extensions in the order in which they appear.</p> <ul style="list-style-type: none"> ▶ Name. The name of the Data Format Extension. ▶ Tag Name. This is the extension's unique ID. ▶ Description. A brief description of the Data Format Extension. For a more comprehensive description, see "Data Format Extension List" on page 901. ▶ Provider. The creator of the Data Format Extension. ▶ Continue Processing. This condition determines what to do if the Data Format Extension successfully deserializes the target data. If the condition is true, VuGen continues to pass the converted data to the next Data Format Extension in the chain. If the condition is false, the chain is terminated.
Headers Chain	A chain of Data Format Extensions that is active on HTTP headers.
Body Chain	A chain of Data Format Extensions that is active on HTTP bodies.
Cookies Chain	A chain of Data Format Extensions that is active on HTTP cookies.
Query String Chain	A chain of Data Format Extensions that is active on query strings.

EJB Code Generation Options Node

Enables you to set the EJB recording options.

To Access	Tools > Recording Options > EJB Options > Code Generation Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Auto Transaction	Automatically marks all EJB methods as transactions. This encloses all methods with Ir.start_transaction and Ir.end_transaction functions. Default value: enabled.
EJB Initialization Method	The method to which the EJB/JNDI initialization properties are written. The available methods are init and action . Default value: init.
Insert Value Check	Automatically insert an Ir.value_check function after each EJB method. This function checks for the expected return value for primitive values and strings.

Flex Code Generation Node

Enables you to set the code generation options for the Flex protocol.

To Access	Tools > Recording Options > Flex > Code Generation
------------------	--

Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
Relevant Tasks	"How to Handling Externalizable Objects in Flex" on page 665

User interface elements are described below:

UI Element (A-Z)	Description
Encode AMF3 using external parser	Attempt to encode externalizable objects using an external parser.
Encode externalizable object with LoadRunner parser	<p>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.</p> <p>To overcome this issue, VuGen generates Custom Request functions containing unparsed AMF3 binary data when it encounters an externalizable object.</p> <p>You can attempt to encode these objects using the LoadRunner parser as well. This decodes all externalizable objects as standard AMF3 objects, generating amf_call functions. However, if objects cannot be parsed, they are not decoded at all so you have to check the script after it is generated to make sure that the parser was effective. Additionally, this option may reduce the stability of code generation.</p> <p>Default value: enabled.</p>
Flex server/ application JAR files location	The location of the external parser and JAR files to be encoded.

General Protocol Node

Enables you to set the script generation preferences by setting the scripting language and options.

To Access	Tools > Recording Options > General > Protocols
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Active Protocols List	A list of the protocols which comprise your multiple protocol script. VuGen lets you modify the protocol list for which to generate code during the recording session. Select the check boxes adjacent to the protocols you want to record in the next recording session. Clear the check boxes adjacent to the protocols you do not want to record in the next recording session.

General Recording Node

Enables you to specify what information to record and which functions to use when generating a Vuser script, by selecting a recording level.

To Access	Tools > Recording Options > General > Recording
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Advanced HTML Dialog Box.
	Opens the Advanced URL Dialog Box.
GUI-based script	Recommended for most applications, including those with JavaScript. Also recommended for PeopleSoft Enterprise and Oracle Web Applications.
HTML-based script	This is the default recording level for Web (HTTP/HTML) Users. It instructs VuGen to record HTML actions in the context of the current Web page. It does not record all resources during the recording session, but downloads them during replay. This options is recommended for browser applications with applets and VB script.
URL-based script	Record all requests and resources from the server. It automatically records every HTTP resource as URL steps (web_url statements), or in the case of forms, as web_submit_data . It does not generate the web_link , web_image , and web_submit_form functions, nor does it record frames. This options is recommended For non-browser applications.

Advanced URL Dialog Box

Enables you to set the advanced options for scripts using the URL recording mode.

To Access	Tools > Recording Options > General > Recording > URL Advanced
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Restores the default settings of this dialog box.
Create concurrent groups for resources after their source HTML page	Records the resources in a concurrent group (enclosed by web_concurrent_start and web_concurrent_end statements) after the URL. Resources include files such as images and js files. If you disable this option, the resources are listed as separate web_url steps, but not marked as a concurrent group.
Enable EUC-Encoded Web Pages	(For Japanese windows only) Instructs VuGen to use EUC encoding. For more information, see "EUC-Encoding (Japanese Windows only)" on page 310.
Use web_custom_request only	Records all HTTP requests as custom requests. VuGen generates a web_custom_request function for all requests, regardless of their content. Recommended for non-browser applications.

Advanced HTML Dialog Box

Enables you to set the advanced options for HTTP-based scripts.

To Access	Tools > Recording Options > General > Recording > HTML Advanced
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
 A rectangular button with a light gray background and a thin black border. The text "Use Defaults" is centered in a dark gray font. The letter "U" is underlined.	Restores the default settings of this dialog box.

UI Element (A-Z)	Description
Non-HTML generated elements	<p>Many Web pages contain non-HTML elements, such as applets, XML, ActiveX elements, or JavaScript. These non-HTML elements usually contain or retrieve their own resources. Using the following options, you can control how VuGen records non HTML-generated elements.</p> <ul style="list-style-type: none"> ▶ Record within the current script step. Does not generate a new function for each of the non HTML-generated resources. It lists all resources as arguments of the relevant functions, such as web_url, web_link, and web_submit_data. The resources, arguments of the Web functions, are indicated by the EXTRARES flag. ▶ Record in separate steps and use concurrent groups. Creates a new function for each one of the non HTML-generated resources and does not include them as items in the page's functions (such as web_url and web_link). All of the web_url functions generated for a resource are placed in a concurrent group (surrounded by web_concurrent_start and web_concurrent_end). ▶ Do not record. Does not record any non-HTML generated resources.

UI Element (A-Z)	Description
Script type	<ul style="list-style-type: none"> ▶ A script describing user actions. Generates functions that correspond directly to the action taken. It creates URL (web_url), link (web_link), image (web_image), and form submission (web_submit_form) functions. The resulting script is very intuitive and resembles a context sensitive recording. ▶ A script containing explicit URL's only. Records all links, images and URLs as web_url statements, or in the case of forms, as web_submit_data. It does not generate the web_link, web_image, and web_submit_form functions. The resulting script is less intuitive. This mode is useful for instances where many links within your site have the same link text. If you record the site using the first option, it records an ordinal (instance) for the link, but if you record using the second option, each link is listed by its URL. This facilitates parameterization and correlation for that step.

General Script Node

Enables you to set the script generation preferences by setting the scripting language and options.

To Access	Tools > Recording Options > General > Script
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Close all AUT processes when recording stops	Automatically closes all of the AUT's (Application Under Test) processes when VuGen stops recording. Default value: disabled.
Correlate arrays	Tracks and correlates arrays of all data types, such as string, structures, numbers, and so on. Default value: enabled.
Correlate large numbers	Correlates long data types such as integers, long integers, 64-bit characters, float, and double. Default value: disabled.
Correlate simple strings	Correlates simple, non-array strings and phrases. Default value: disabled.
Correlate small numbers	Correlates short data types such as bytes, characters, and short integers. Default value: disabled.
Correlate structures	Tracks and correlates complex structures. Default value: enabled.
Declare primitives as locals	Declares primitive value variables as local variables rather than class variables (C, C#, and .NET only). Default value: enabled.
Explicit variant declaration	Declares variant types explicitly in order to handle ByRef variants (Visual Basic for Applications only). Default value: enabled.
Generate fixed think time after end transaction	Adds a fixed think time, in seconds, after the end of each transaction. When you enable this option, you can specify a value for the think time. Default value: disabled, 3 seconds when enabled.
Generate recorded events log	Generates a log of all events that took place during recording. Default value: disabled.

UI Element (A-Z)	Description
Generate think time greater than threshold	<p>Uses a threshold value for think time. If the recorded think time is less than the threshold, VuGen does not generate a think time statement. You also specify the threshold value. The default values is 3—if the think time is less than 3 seconds, VuGen does not generate think time statements. If you disable this option, VuGen will not generate any think times.</p> <p>Default value: enabled, 3 seconds.</p>
Insert output parameters values	<p>Inserts output parameter values after each call (C, C#, and .NET only).</p> <p>Default value: disabled.</p>
Insert post-invocation info	<p>Insert informative logging messages after each message invocation (non-C only).</p> <p>Default value: enabled.</p>
Insert pre-invocation info	<p>Insert informative logging messages before each message invocation (non-C only).</p> <p>Default value: enabled.</p>
Maximum number of lines in action file	<p>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).</p> <p>Default value: disabled.</p>
Replace long strings with parameter	<p>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:</p> <p>const char <paramName_uniqueID> = "string".</p> <p>This option allows you to have a more readable script. It does not effect the performance of the script.</p> <p>Default value: enabled.</p>

UI Element (A-Z)	Description
Reuse variables for primitive return values	Reuse the same variables for primitives received from method calls. This overrides the Declare primitives as locals setting . Default value: enabled.
Track processes created as COM local servers	Track the activity of the recorded application if one of its sub-processes was created as a COM local server (C and COM only). Default value: enabled.
Use full type names	Use the full type name when declaring a new variable (C# and .NET only). Default value: disabled.
Use helpers for arrays	Use helper functions to extract components in variant arrays (Java and VB Scripting only). Default value: disabled.
Use helpers for objects	Use helper functions to extract object references from variants when passed as function arguments (Java and VB Scripting only). Default value: disabled.

GUI Properties Advanced Node

Enables you to set advanced recording options for Click and Script Users.

To Access	Tools > Recording Options > GUI Properties > Advanced
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Code Generation Settings	For information about this section, see "Code Generation Settings Properties" on page 358.
Recording Settings	For information about this section, see "Recording Settings Properties" on page 357.

Recording Settings Properties

UI Element (A-Z)	Description
Generate snapshots for AJAX steps	Enables generation of snapshots for AJAX steps. Enabling this option can result in errors during recording. Default value: disabled.
Record 'click' by mouse events	Records mouse clicks by capturing mouse events instead of capturing the click() method. Enable when the recorded application uses the DOM click() method, to prevent the generation of multiple functions for the same user action. Default value: enabled.
Record rendering-related property values	Records the values of the rendering-related properties of DOM objects (for example, offsetTop), so that they can be used during replay. Note that this may significantly decrease the replay speed. Default value: disabled.
Record socket level data	Enables the recording of socket level data. If you disable this option you will need to manually add the starting URL before recording. In addition, you will be unable to regenerate the script on an HTML level. Default value: enabled.

Code Generation Settings Properties

UI Element (A-Z)	Description
Enable automatic browser title verification	Enables automatic browser title verification. Default value: disabled.
Enable generation of out-of-context steps	Creates a URL-based script for ActiveX controls and Java applets, so that they will be replayed. Since these functions are not part of the native recording, they are referred to as out-of-context recording. Default value: disabled.
Perform a title verification for	<ul style="list-style-type: none"> ▶ 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.

GUI Properties Web Event Configuration Node

Enables you to set the level of detail recorded in a script (web event recording).

To Access	Tools > Recording Options > GUI Properties > Web Event Configuration
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Basic Event Configuration Level	<ul style="list-style-type: none"> ▶ Always records click events on standard Web objects such as images, buttons, and radio buttons. ▶ Always records the submit event within forms. ▶ Records click events on other objects with a handler or behavior connected. ▶ Records the mouseover event on images and image maps only if the event following the mouseover is performed on the same object.
Custom Settings	Opens the Custom Web Event Recording Configuration Dialog Box, where you can customize the event recording configuration.
High Event Configuration Level	In addition to the objects recorded in the Medium level, it records mouseover, mousedown, and double-click events on objects with handlers or behaviors attached.
Medium Event Configuration Level	In addition to the objects recorded in the Basic level, it records click events on the <DIV>, , and <TD> HTML tag objects.

Custom Web Event Recording Configuration Dialog Box

Enables you to customize the level of web event recording.

To Access	Tools > Recording Options > GUI Properties > Web Event Configuration > Custom Settings
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
<Object List>	A list of the web objects. Each web object can be customized according to the other settings in this dialog box.
<Object Menu>	<ul style="list-style-type: none"> ▶ Add. Adds a new HTML tag object to the object list. Type in the name of the tag. ▶ Delete. Deletes an object from the object list.
Event Menu	<ul style="list-style-type: none"> ▶ Add. Adds an event to the Event Name column of this object. ▶ Delete. Deletes an event from the Event Name column of this object.
Event Name	A list of events associated with the object.
File Menu	<ul style="list-style-type: none"> ▶ Load Configuration. Loads a previously created custom configuration. ▶ Save Configuration As. Saves the current configuration.
Listen	<p>The criteria which determines when VuGen listens for an event.</p> <ul style="list-style-type: none"> ▶ Always. Always listen to the event. ▶ If Handler. Listens to the event if a handler is attached to it. A handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs. ▶ If Behavior. Listens to the event if a DHTML behavior is attached to it. A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior. ▶ If Handler or Behavior. Listens to the event if either a handler or a behavior is attached to it. ▶ Never. Never listens to the event. <p>For more information, see "Tips for Working with Event Listening and Recording" on page 316.</p>

UI Element (A-Z)	Description
Record	<p>The criteria which determines when VuGen records an event.</p> <ul style="list-style-type: none"> ▶ Enabled. Records the event each time it occurs on the object as long as VuGen listens to the event on the selected object, or on another object to which the event bubbles. Bubbling is the process whereby, when an event occurs on a child object, the event can travel up the chain of hierarchy within the HTML code until it encounters an event handler to process the event. ▶ Disabled. Does not record the specified event and ignores event bubbling where applicable. ▶ Enabled on next event. Same as Enabled, except that it records the event only if the subsequent event occurs on the same object. For example, suppose a mouseover behavior modifies an image link. You may not want to record the mouseover event each time you happen to move the mouse over this image. Because only the image that is displayed after the mouseover event enables the link event, however, it is essential that the mouseover event is recorded before a click event on the same object. <p>For more information, see "Tips for Working with Event Listening and Recording" on page 316.</p>
Reset Settings	Resets the custom settings to the settings of your choice: basic , medium , or high .

HTTP Advanced Node

Enables you to customize the code generation settings in the area of think time, resetting contexts, saving snapshots, and the generation of `web_reg_find` functions.

To Access	Tools > Recording Options > HTTP Properties > Advanced
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ▶ Some options within this node are not available in when using a multi-protocol script.

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Headers Dialog Box.
	Opens the Content Type Filters Dialog Box.
	Opens the Non-Resources Dialog Box.
Add comment to script for HTTP errors while recording	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.
Generate web_reg_find functions for page titles	<p>Generates <code>web_reg_find</code> functions for all HTML page titles. VuGen adds the string from the page's title tag and uses it as an argument for <code>web_reg_find</code>.</p> <ul style="list-style-type: none"> ▶ Generate web_reg_find functions for sub-frames. Generates <code>web_reg_find</code> functions for page titles in all sub-frames of the recorded page. <p>Note: This option is only available for Web and Oracle NCA protocols</p>

UI Element (A-Z)	Description
Record script using earlier recording engine	Record using the single-protocol recording engine. By default, for Web (HTTP/HTML) Vusers, VuGen uses the multi-protocol recording engine for all recordings even if you are only recording a single protocol.
Record think time	Records the think times and generate lr_think_time functions. You can also set a Think-time Threshold value to only generate lr_think_time functions when the actual think-time is greater than the threshold. Note: This option is only available for Wireless Protocols scripts.
Reset context for each action	Resets all HTTP contexts between actions. Resetting contexts allows the Vuser to more accurately emulate a new user beginning a browsing session. This option resets the HTML context, so that a context-less function is always recorded in the beginning of the action. It also clears the cache and resets the user names and passwords. Note: This option is only available for Web and Oracle NCA protocols

UI Element (A-Z)	Description
Save snapshot resources locally	Saves a local copy of the snapshot resources during record and replay, thereby creating snapshots more accurately and displaying them quicker.
Support charset	<ul style="list-style-type: none"> ▶ UTF-8. Enables support for UTF-8 encoding. This instructs VuGen to convert non-ASCII UTF-8 characters to the encoding of your locale's machine in order to display them properly in VuGen. You should enable this option only on non-English UTF-8 encoded pages. The recorded site's language must match the operating system language. You cannot record non-English Web pages with different encodings (for example, UTF-8 together with ISO-8859-1 or shift_jis) within the same script. ▶ EUC-JP. If you are using Japanese Windows, select this option to enable support for Web sites that use EUC-JP character encoding. This instructs VuGen to convert EUC-JP strings to the encoding of your locale's machine in order to display them properly in VuGen. VuGen converts all EUC-JP (Japanese UNIX) strings to the SJIS (Japanese Windows) encoding of your locale's machine, and adds a web_sjis_to_euc_param function to the script. (Kanji only)

Headers Dialog Box

Enables you to automatically send additional HTTP headers with every HTTP request submitted to the server.

To Access	Tools > Recording Options > HTTP > Advanced > Headers
Important Information	<ul style="list-style-type: none"> ▶ This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ▶ The following standard headers are considered risky: Authorization, Connection, Content-Length, Cookie, Host, If-Modified-Since, Proxy-Authenticate, Proxy-Authorization, Proxy-Connection, Referer, and WWW-Authenticate. They are not recorded unless selected in the Header list.

User interface elements are described below:

UI Element (A-Z)	Description
	Plus. Adds a new entry.
	Minus. Deletes an entry.
	Restores the current list to the default values and entries.
	Restores all lists to the default values and entries.
<Drop-down menu>	Controls the options for this dialog box: <ul style="list-style-type: none"> ▶ Do not record headers. ▶ Record headers in list. ▶ Record headers not in list.
<Header list>	List of headers which may or may not be recorded. The lists vary depending on which drop-down item is selected. Each item can be selected or deselected using its individual checkbox.

Content Type Filters Dialog Box

Enables you to filter content types for your recorded script. You can specify the type of the content you want to record or exclude from your script.

To Access	Tools > Recording Options > HTTP > Advanced > Content Types
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Plus. Adds a new entry.
	Minus. Deletes an entry.
	Restores the current list to the default values and entries.
	Restores all lists to the default values and entries.
<Drop-down menu>	Controls the options for this dialog box: <ul style="list-style-type: none"> ▶ Do not filter content types. ▶ Filter content types in list. ▶ Filter content types not in list.
<Header list>	List of content types which may or may not be filtered. The lists vary depending on which drop-down item is selected. Each item can be selected or deselected using its individual checkbox.

Non-Resources Dialog Box

When you record a script, VuGen indicates whether or not it will retrieve the resource during replay using the Resource attribute in the **web_url** function. If the Resource attribute is set to 0, the resource is retrieved during script execution. If the Resource attribute is set to 1, the Vuser skips the resource type.

You can exclude specific content types from being handled as resources. For example, you can indicate to VuGen that **gif** type resources should not be handled as a resource and therefore be downloaded unconditionally.

To Access	Tools > Recording Options > HTTP > Advanced > Non-Resources
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

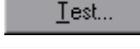
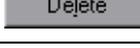
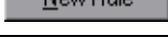
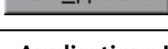
UI Element (A-Z)	Description
	Add. Adds a new entry to the list.
	Remove. Deletes an entry from the list.
	Restores the default list.
<Non-Resource Content Type list>	List of items which should not be recorded as resources. Each item can be selected or deselected using its individual checkbox.

HTTP Correlation Node

Enables you to create correlation rules that automatically correlate statements during recording.

To Access	Tools > Recording Options > HTTP Properties > Correlation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Imports a correlation file containing rule information.
	Exports a correlation file containing rule information.
	Opens the Token Substitution Testpad Dialog Box.
	Delete the selected rule or application.
	Opens the New Rule Pane.
	Adds a new application to the Application List.
<Applications List>	The list of applications and their rules.
Add comment to script	Adds descriptive comments to the correlation steps.
Enable correlation during recording	Enables correlation during recording according to the specified correlation rules.

 **New Rule Pane**

Enables you to define a new custom rule

To Access	Tools > Recording Options > HTTP > Correlation > New Rule
Important Information	This pane is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Advanced Correlation Properties Dialog Box.
Action	<p>Specify the type of action for the rule from the following options:</p> <ul style="list-style-type: none"> ▶ Search for Parameters in all of the Body Text. Searches the entire body—not just links, form actions or cookies. It searches the text for a match using the borders that you specify. ▶ Search for parameters in links and form actions. Searches within links and forms' actions for the text to parameterize. This method is for application servers where you know the context rules. You define a left boundary, a right boundary, an alternate right boundary, and an instance of the left boundary within the current link. ▶ Search for Parameters from cookie headers. Similar to the previous rule, except that the value is extracted from cookie text (exactly as it appears in the recording log) instead of from a link or form action. ▶ Parameterize form field value. Saves the named form field value to a parameter. It creates a parameter and places it in the script before the form's action step. For this option, you need to specify the field name. ▶ Text to enter a web_reg_add_cookie function by method inserts a web_reg_add_cookie function if it detects a certain string in the buffer. It only adds the function for those cookies with the specified prefix. For this option, you need to specify the search text and the cookie prefix.
Field name	The field name. This field must be completed when creating an action of type Parameterize form field value .
Left boundary	The left-most boundary where the rule will apply.
Match Case	Matches the case when looking for boundaries.

UI Element (A-Z)	Description
Parameter prefix	Uses a prefix in all automatically generated parameters based on this rule. Prefixes prevent you from overwriting existing user parameters. In addition, prefixes allow you to recognize the parameter in your script. For example, in Siebel Web, one of the built-in rules searches for the Siebel_row_id prefix.
Right boundary	The right-most boundary where the rule will apply. Use the drop-down menu to define this boundary as either the end of a string, a newline character, or a user-defined text.
Use '#' for any digit	Replaces all digits with a hash sign. The hash signs serve as wildcard, allowing you to find text strings with any digit. Example: If you enable this option and specify HP### as the left boundary, HP193 and HP284 will be valid matches.

Advanced Correlation Properties Dialog Box

Enables you to set the advanced options for correlation rules.

To Access	Tools > Recording Options > HTTP > Correlation > New Rule > Advanced
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Alternate right boundary	Alternative criteria for the right boundary, if the previously specified boundary is not found. Select one of the following options: User-defined Text, Newline Character, End Of Page.
Always create new parameter	Creates a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance.
Left boundary instance	The number of occurrences of the left boundary in order for it to be considered a match.
Length	The length of the string, starting with the offset, to save to the parameter. If this is not specified, the parameter continues until the end of the found value.
Offset	The offset of the string within the found value.
Replace with parameter only for exact matches	Replaces a value with a parameter only when the text exactly matches the found value.
Reverse search	Performs a backwards search.

Token Substitution Testpad Dialog Box

Enables you to test correlation rules before applying them

To Access	Tools > Recording Options > HTTP > Correlation > Test
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Runs the test.
Applied rules	A list of the rules that were applied during the test.
Source string for substitution	Enter the source string for substitution.
Substitution Result	The results of the test.

Java Classpath Node

Enables you to specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper recording.

To Access	Tools > Recording Options > Java Environment Settings > Classpath
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Down Arrow. Moves a classpath entry down the list.
	Up Arrow. Moves a classpath entry up the list.
	Add Classpath. Adds a new line to the classpath list.

UI Element (A-Z)	Description
	Delete. Permanently removes a classpath.
Classpath Entries List	A list of classpath entries.

Java VM Node

Enables you to indicate additional parameters to use when recording Java applications.

To Access	Tools > Recording Options > Java Environment Settings > Java VM
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Additional VM Parameters	List the Java command line parameters here. These parameters may be any Java VM argument. The common arguments are the debug flag (-verbose) or memory settings (-ms , -mx). In addition, you may also pass properties to Java applications in the form of a -D flag. For more information about the Java VM flags, see the JVM documentation.
Prepend CLASSPATH to -Xbootclasspath parameter	Instructs VuGen to add the Classpath before the Xbootclasspath (prepend the string).

UI Element (A-Z)	Description
Use classic Java VM	Instructs VuGen to use the classic version of VM (for example, not Sun's Java HotSpot).
Use the specified Additional VM Parameters during replay	Instructs VuGen to use the same Additional VM parameters in replay.

Microsoft .NET Filters Node

Enables you to set the recording options for the Microsoft .NET protocol.

To Access	Tools > Recording Options > Microsoft .NET > Filters
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
See Also	"Microsoft .NET Filters Overview" on page 757 "Microsoft .NET Filters - Advanced" on page 759 "Guidelines for Setting Microsoft .NET Filters" on page 760

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Create a New Filter dialog box, enabling you to create a new filter. For more details, see "Create a New Filter Dialog Box" on page 376.
	Opens the Filter Manager, allowing you to view and modify all Microsoft .NET protocol filters.
Custom Filter	Shows the filters that you created earlier on the current machine.

UI Element (A-Z)	Description
Environment Filter	Lists the available environment filters: .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation of Framework 3.0).
New Filter	Indicates that you want to create a new filter.

Create a New Filter Dialog Box

This dialog box enables you to create a new filter for Microsoft .NET scripts.

To access	Tools > Recording Options > Microsoft .NET > Filters > New Filter > Create
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
Based on a custom filter	Create a new filter based on a custom filter. Use the drop-down menu to select the custom filter.
Based on an environment filter	Create a new filter based on an environment filter. Use the check boxes next to the environment filters to indicate which environment filters to base the filter on.
Start with an empty filter	Create a new filter that is not based on a pre-existing filter.

Filter Manager

This dialog box enables you to create and edit Microsoft .NET filters

To access	Tools > Recording Options > Microsoft .NET > Filters > Environment / Custom Filter > Filter Manager
See also	"Microsoft .NET Filters Overview" on page 757 "Microsoft .NET Filters - Advanced" on page 759 "Guidelines for Setting Microsoft .NET Filters" on page 760

User interface elements are described below:

UI Elements (A-Z)	Description
	Includes the selected element. If you manually include a parent node, the Filter Manager includes the child elements below it, provided that no other rule exists. For example, if you include a class, it will include all its methods unless you specifically excluded a method.
	Excludes the selected element. The child elements are also excluded unless they were included by another rule. By default, when you exclude a class, the Filter Manager applies the Exclude attribute to the class, but it allows the recording engine to record activity within the methods of the excluded class. When you exclude a method, however, the Filter Manager applies Totally Exclude, preventing the recording engine from recording any activity within the methods of the excluded class. Advanced users can modify these setting in the filter file. For more information, see "Advanced Information About Filter Files" on page 826.

UI Elements (A-Z)	Description
	<p>Removes the manual inclusion or exclusion rule. In this case, the element may be impacted by other parent elements.</p> <p>The inclusion and exclusion rules have the following properties:</p> <ul style="list-style-type: none"> ▶ 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 <code>.cctor()</code> and <code>Dispose(bool)</code>, do not follow the standard hierarchal rules. <p>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.</p> <p>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.</p> <p>For tips about including and excluding elements in the filter, see "Determining which Elements to Include or Exclude" on page 761.</p>
	<p>Navigates to the previous or next tree node visited by the user.</p>

UI Elements (A-Z)	Description
Impact Log	<p>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.</p> <p>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.</p> <p>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.</p>
< Filter Manager Tree >	<p>The Filter Manager tree uses symbols to illustrate the elements and their status. For details about each of the icons, see the table below.</p> <ul style="list-style-type: none"> ▶ Element icons represent the type of element—assembly, namespace, class, method, structure, property, events, or interfaces. ▶ A check mark or X adjacent to the element icon, indicates whether or not the element is included or excluded. ▶ A bold element indicates that it was explicitly included or excluded. This may be a result of being manually included or excluded by the user or by a pre-defined rule in the environment filter. If you reset a bold node, it returns to its original, non-bold state.
Add Reference	<p>Opens the Add Reference dialog box with a list of .NET Framework components or assemblies in the Public Assemblies folder. For more information, see "Add Reference Dialog Box" on page 381.</p>
Delete	<p>Deletes the selected custom filter. The Filter Manager prompts you for a confirmation.</p>
New	<p>Opens the Create a New Filter dialog box, in which you create an empty filter or a new filter based on an existing one. For more information, see "Create a New Filter Dialog Box" on page 376.</p>

UI Elements (A-Z)	Description
Remove Reference	Removes the assembly that is selected in the Filter Manager and all of the elements associated with it. The Filter Manager prompts you for a confirmation.
Save	Saves the changes you made to filter.
View Impact Log	Opens the Impact log for the selected filter. The Impact log shows which nodes in the tree were affected by recent actions. For more information, see "Viewing an Impact Log" on page 825.

The following table shows the Filter Manager Tree 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

Add Reference Dialog Box

This dialog box enables you to add references to Microsoft .NET filters.

To access	Tools > Recording Options > Microsoft .NET > Filters > Environment / Custom Filter > Filter Manager > Add Reference
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
<Component List>	<p>A list of .NET Framework components or assemblies in the Public Assemblies folder.</p> <ul style="list-style-type: none"> ▶ To add one of the listed items, select it and click Select. You can select multiple components using ctrl-click. The bottom pane shows the selected references. ▶ To add an assembly that is not in the list, click Browse and locate the reference on your file system or network.
<Selected Component List>	<p>The list of selected components. The Type column indicates .NET for a component from the Public Assemblies folder and File for a component that was added by selecting Browse.</p> <ul style="list-style-type: none"> ▶ To clear an item from the list, select it in the bottom pane and click Remove.

Microsoft .NET Recording Node

Enables you to set the recording options for the Microsoft .NET protocol.

To Access	Tools > Recording Options > Microsoft .NET> Recording
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Code Generation	<p>Allow you to indicate whether to show warnings, a stack trace, or all event subscriptions during code generation.</p> <ul style="list-style-type: none"> ▶ Show Warnings. Shows warning messages that are issued during the code generation process. ▶ Show Stack Trace. Shows the recorded stack trace if it is available. ▶ Show All Event Subscriptions. Generates code for all event subscriptions that were recorded. If this option is disabled, VuGen will only generate code for events in which both the publisher (the object which invokes the event) and the subscriber (the object informed of the event) are included in the filter. <p>Default value: disabled.</p>
Debug Options	<p>Enables you to trace the stack and specify its size.</p> <ul style="list-style-type: none"> ▶ 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. <p>Default value: disabled.</p> <ul style="list-style-type: none"> ▶ Stack Trace Limit. The maximum number of calls to be stored in the stack. If the number of calls exceeds the limit, VuGen truncates it. <p>Default value: 20 calls.</p>
Filters	<ul style="list-style-type: none"> ▶ Ignore all assemblies by default. Ignores all assemblies that are not explicitly included by the selected filter. If you disable this option, VuGen looks for a matching filter rule for all assemblies loaded during the recording.

UI Element (A-Z)	Description
<p>Logging</p>	<p>The Logging options let you set the level of detail that is recorded in the recording log file.</p> <ul style="list-style-type: none"> ➤ Log severity. Sets the level of logging to Errors Only (default), or Debug. The severity setting applies for all the logs that you enable below. You should always use the Errors Only log unless specifically instructed to do otherwise by HP support, since detailed logging may significantly increase the recording time. ➤ Instrumentation Log. Logs messages related to the instrumentation process. Default value: enabled. ➤ Recording Log. Logs messages issued during recording. Default value: enabled. ➤ Code Generation Log. Logs messages issued during the code generation stage. Default value: enabled.

UI Element (A-Z)	Description
Remote Objects	For information about this property, see "Remote Objects Property" on page 386.
Serialization	<ul style="list-style-type: none"> ▶ 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. <p>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.</p>

 **Remote Objects Property**

User interface elements are described below:

UI Element (A-Z)	Description
Record in-process objects	Records activity between the client and server when the server is hosted in the same process as the client. Since the actions are not true client/server traffic, it is usually not of interest. When in-process methods are relevant, for example, in certain Enterprise Service applications, you can enable this option to capture them. Default value: disabled.

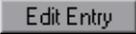
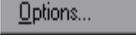
UI Element (A-Z)	Description
Asynchronous calls	<p>Specifies how VuGen should handle asynchronous calls on remote objects and their callback methods</p> <ul style="list-style-type: none"> ▶ Call original callbacks by default. Uses the recorded application's original callback when generating and replaying the script. If the callback method is explicitly excluded by a filter, the callback will be excluded even if you enable this option. ▶ Generate asynchronous callbacks. This option defines how VuGen will handle callbacks when the original callbacks are not recorded. <p>For more information, see "Asynchronous Calls" on page 752.</p>
WCF duplex binding	<ul style="list-style-type: none"> ▶ Generate dummy callback handler. Replaces the original callback in duplex communication with a dummy callback, performing the following actions: <ul style="list-style-type: none"> ▶ Store arguments. When the server calls the handler during replay, it saves the method arguments to a key-value in memory map. ▶ Synchronize replay. It stops the script execution until the next response arrives. VuGen places the synchronization at the point that the callback occurred during recording. This is represented in the script by a warning: ▶ Generate unique client base address. If your application employs dual HTTP Binding, since HTTP is inherently not a duplex protocol, the framework uses a standard port to receive response data being passed to the callback. When you attempt to run multiple instances of your application, you may be unable to do so using the same port number. This option replaces the original client base address's port number with a unique port. <p>For background information about WCF duplex binding, see "Recording WCF Duplex Communication" on page 746.</p>

Network Port Mapping Node

Enables you to set the port mapping recording options.

To Access	Tools > Recording Options > Network > Port Mapping
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Server Entry dialog box, allowing you to add a new mapping. For user interface details, see "Server Entry Dialog Box" on page 389.
	Opens the Server Entry dialog box, allowing you to edit the selected entry. For user interface details, see "Server Entry Dialog Box" on page 389.
	Opens the Advanced Settings dialog box to enable auto-detection of the communication protocol and SSL level. For user interface details, see "Advanced Port Mapping Settings Dialog Box" on page 392.
<Port Mapping list>	A list of the port mappings. You can temporarily disable any entry by clearing the check box adjacent to it. When you disable an entry, VuGen ignores all traffic to that server:port combination. You should disable the port entry when the data is irrelevant or if the protocol is not supported.

UI Element (A-Z)	Description
Capture level	<p>The level of data to capture (relevant only for HTTP based protocols):</p> <ul style="list-style-type: none"> ▶ Socket level data. Capture data using trapping on the socket level only. Port mappings apply in this case (default). ▶ WinINet level data. Capture data using hooks on the WinINet.dll API used by certain HTTP applications. The most common application that uses these hooks is Internet Explorer. Port mappings are not relevant for this level. ▶ Socket level and WinINet level data. Captures data using both mechanisms. WinINet level sends information for applications that use WinINet.dll. Socket level sends data only if it determines that it did not originate from WinINet.dll. Port mapping applies to data that did not originate from WinINet.dll.
Network-level server address mappings for:	Specifies the mappings per protocol. For example, to show only the FTP mappings, select FTP.

Server Entry Dialog Box

Enables you to define a server from the server list in the network port mapping node.

To Access	Tools > Recording Options > Network > Port Mapping > New Entry / Edit Entry
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
<p>Allow forwarding to target server from local port</p>	<p>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. You configure VuGen to intercept the traffic from the problematic client machine, and pass it on to the server. In this way, VuGen can process the data and generate code for the actions.</p> <p>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.</p> <p>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.</p>
<p>Connection Type</p>	<p>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.</p>

UI Element (A-Z)	Description
Port	<p>The port of the target server for which this entry applies. Entering 0 specifies all ports.</p> <p>If you do not specify all of the port and server names, VuGen uses the following priorities in assigning data to a service:</p> <ul style="list-style-type: none"> ➤ Priority 1: port and server specified ➤ Priority 2: port not specified, server specified ➤ Priority 3: port specified, server not specified ➤ Priority 4: port and server not specified <p>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.</p> <ul style="list-style-type: none"> ➤ Forced mapping. If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol to record that communication, even though the actual communication may be HTTP. In this instance, the Vuser script might be empty.
Record Type	The type of recording—directly or through a proxy server.
Service ID	A protocol or service name used by the recorder to identify the type of connection (i.e. HTTP, FTP, and so on). You can also specify a new name. The name may not exceed 8 characters.
Service Type	The type of service, currently set to TCP.
SSL Cipher	The preferred SSL cipher to use when connecting with a remote secure server.

UI Element (A-Z)	Description
SSL Version	The preferred SSL version to use when communicating with the client application and the server. Default value: SSL 2/3. However some services require SSL 3.0 only or SSL 2.0 only. Some new wireless applications require TLS 1.0—a different security algorithm.
Target Server	The IP address or host name of the target server for which this entry applies. Default value: All Servers.
Use specified client-side certificate	The default client-side certificate to use when connecting to a remote server. Specify or browse for a certificate file in txt , crt , or pem format, and supply a password.
Use specified proxy-server certificate	The default server certificate to present to client applications that request a server certificate. Specify or browse for a certificate file in txt , crt , or pem format, and supply a password. Click Test SSL to check the authentication information against the server.

 **Advanced Port Mapping Settings Dialog Box**

Enables you to set the advanced port mapping settings. For more information, see "Port Mapping Auto Detection" on page 308.

To Access	Tools > Recording Options > Network > Port Mapping > Options
Important Information	This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Enable auto detection of SOCKET based communication	Automatically detects the type of communication. If required, raise the maximum number of transitions, one at a time until VuGen succeeds in detecting the protocol. You can also gradually increase the maximum buffer size by 1024 bytes (1 KB) at a time until VuGen succeeds in detecting the protocol. This allows VuGen to review a larger amount of data in order to find a signature.
Enable auto SSL detection	Automatically detects SSL communication. Specify the version and default cipher that you want to detect. Note that this only applies to port mappings that were defined as auto in the Connection type box, or not defined at all. If a server, port, or server:port combination was defined as either Plain or SSL , then auto SSL detection does not apply.
Log Level	Sets the logging level for the automatic socket detection.

RDP Code Generation - Advanced Node

Enables you to control the way VuGen creates an RDP script. Only advanced users are advised to modify these settings.

To Access	Tools > Recording Options > RDP > Code Generation - Adv
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Correlate clipboard parameters	Replaces the recorded clipboard text sent by the user with the correlated parameter containing the same text as received from the server.
Double-click timeout (msec)	The maximum time (in milliseconds) between two consecutive mouse button clicks to be considered a double-click. Default value: 500 milliseconds.
Prefix for clipboard parameters	The prefix for clipboard parameters generated in the current script. This is useful when merging scripts, allowing you to specify a different prefix for each script. Default value: <code>ClipboardDataParam_</code> .
Prefix for snapshot names	The prefix for snapshot file names generated in the current script. This is useful when merging scripts—you can specify a different prefix for each script. Default value: <code>snapshot_</code> .

RDP Code Generation - Agent Node

Enables you to control the way the agent for Microsoft Agent for Terminal Server functions with VuGen during recording.

To Access	Tools > Recording Options > RDP > Code Generation - Agent
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Enable RDP agent log	<p>Enables the RDP agent log.</p> <ul style="list-style-type: none"> ▶ RDP agent log detail level. Configures the level of detail generated in the RDP agent log with Standard being the lowest level of detail and Extended Debug being the highest level of detail. ▶ RDP agent log destination. Configures the destination of the RDP agent log data. File saves the log messages only on the remote server side. Stream sends the log messages to the Vugen machine. FileAndStream sends the log messages to both destinations. ▶ RDP agent log folder. The folder path on the remote server that the RDP agent log file will be generated in.
Use RDP agent	<p>Generates script using information gathered by the RDP agent during the recording session. The LoadRunner RDP agent must be installed on the server.</p>

RDP Code Generation - Basic Node

Enables you to control the way VuGen creates a script—the level of detail, triggers, and timeouts.

To Access	Tools > Recording Options > RDP > Code Generation - Basic
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Always generate connection name	<p>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.</p> <p>Default value: disabled.</p>
Automatic generation of synchronization points	<p>Synchronization points allow the script to pause in the replay while waiting for a window or dialog to pop-up, or some other control to fulfil a certain condition. This option automatically generates sync_on_image functions before mouse clicks and drags (enabled by default). The Sync radius is the distance from the mouse operation to the sides of the rectangle which defines the synchronization area. The default is 20 pixels. Select one of the following options:</p> <ul style="list-style-type: none"> ▶ None. No synchronization points are automatically added. ▶ Rectangular. Creates synchronization points as rectangular boxes centered around the click or drag location. ▶ Enhanced. Creates synchronization points designed to select only the desired location (e.g. a button) and to react to changes in the UI (e.g. the button moves). If a synchronization region is not recognized, the rectangular synchronization settings are used.
Generate mouse movement calls	<p>Generates rdp_mouse_move calls in the script. When enabled, this option significantly increases the script size.</p> <p>Default value: disabled.</p>
Generate raw keyboard calls	<p>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.</p> <p>Default value: disabled.</p>

UI Element (A-Z)	Description
Generate raw mouse calls	<p>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.</p> <p>Default value: disabled.</p>
Script generation level	<p>The level of the script and the type of API functions to use when generating the script.</p> <ul style="list-style-type: none"> ▶ High. Generate high level scripts. Keyboard events are translated to rdp_type calls. Two consecutive mouse clicks with the same coordinates are translated as a double-click. ▶ Low. Generate low level scripts. Key up/down events are translated into rdp_key events. Modifier keys (Alt, Ctrl, Shift) are used as a KeyModifier parameter for other functions. Mouse up/down/ move events are translated to mouse click/drag events. ▶ Raw. Generates a script on a raw level, by extracting input events from network buffers and generating calls in their simplest form: key up/down, mouse up/down/ move. The KeyModifier parameter is not used.

RDP Login Node

Enables you to set the RDP login recording options.

To Access	Tools > Recording Options > RDP > Login
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Run RDP client application	Connects to the terminal server by running the Terminal Services client.
Use custom connection file	Connects to the terminal server by using an existing connection file. The file should have an *.rdp extension. You can browse for the file on your file system or network.
Use default connection file	Connects to the terminal server by using the Default.rdp file in your document's directory.

Recording Properties Corba Options Node

Enables you to set the CORBA specific recording properties and several callback options.

To Access	Tools > Recording Options > Recording Properties > Corba Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Record Callback Connection	Instructs VuGen to generate a connect statement for the connection to the ORB, for each callback object. Default value: disabled.
Record DLL only	Instructs VuGen to record only on a DLL level. Default value: disabled.

UI Element (A-Z)	Description
Record Properties	Instructs VuGen to record system and custom properties related to the protocol. Default value: enabled.
Resolve CORBA Objects	When correlation fails to resolve a CORBA object, recreate it using its binary data. Default value: disabled.
Show IDL Constructs	Displays the IDL construct that is used when passed as a parameter to a CORBA invocation. Default value: enabled.
Use local vendor classes	Use local vendor classes and add the <code>srv</code> folder to the BOOT classpath. If you disable this option, VuGen uses network classes and adds the script's classes to the classpath. Default value: enabled.
Vendor	The CORBA vendors: Inprise Visibroker , Iona OrbixWeb , or Bea Weblogic .

Recording Properties Correlation Options Node

Allows you to enable automatic correlation, and control its depth.

To Access	Tools > Recording Options > Recording Properties > Correlation Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Advanced Correlation	Enables correlation on complex objects such as arrays and CORBA container constructs and arrays. This type of correlation is also known as deep correlation. Default value: enabled.
Correlate Collection Type	Correlates objects from the Collection class for JDK 1.2 and higher. Default value: disabled.
Correlate String Arrays	Correlate strings within string arrays during recording. If disabled, strings within arrays are not correlated and the actual values are placed in the script. Default value: enabled.
Correlate Strings	Correlate strings in script during recording. If disabled, the actual recorded values are included in the script between quotation marks and all other correlation options are ignored Default value: disabled.
Correlation Level	Indicates the level of deep correlation, the number of inner containers to be scanned. Default value: 15.

Recording Properties Log Options Node

Enables you to determine the level of debug information generated during recording.

To Access	Tools > Recording Options > Recording Properties > Log Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Class Dumping	Dumps all of the loaded classes to the script directory. Default value: disabled.
Digest Calculation	Generate a digest of all recorded objects. Default value: disabled. <ul style="list-style-type: none"> ▶ Exclude from Digest. A list of objects not to be included in the digest calculation. Syntax: java.lang.Object class format, delimiter = ","
Log Level	The level of recording log to generate: <ul style="list-style-type: none"> ▶ None. No log file is created ▶ Brief. Generates a standard recording log and output redirection ▶ Detailed. Generates a detailed log for methods, arguments, and return values. ▶ Debug. Records hooking and recording debug information, along with all of the above.
Synchronize Threads	For multi-threaded applications, instructs VuGen to synchronize between the different threads. Default value: disabled.

Recording Properties Recorder Options Node

Enables you to set the Java protocol to record as well as other protocol specific recording options.

To Access	Tools > Recording Options > Recording Properties > Recorder Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Byte Array Format	The format of byte arrays in a script: Regular , Unfolded Serialized Objects , or Folded Serialized Objects . Use one of the serialized object options when recording very long byte arrays. Default value: Regular.
Bytes as Characters	Displays readable characters as characters with the necessary casting—not in byte or hexadecimal form. Default value: enabled.
Comment Lines Containing	Comment out all lines in the script containing one of the specified strings. To specify multiple strings, separate the entries with commas. Default value: Any line with a string containing <undefined> will be commented out.
Extensions List	A comma separated list of all supported extensions. Each extension has its own hooks file. Default value: JNDI.
Insert Functional Check	Inserts verification code that compares the return value received during replay, to the expected return value generated during recording. This option only applies to primitive return values. Default value: disabled.
Load Parent Class Before Class	Change the loading order so that parent classes are loaded before child classes. This helps identify hooking for trees with deep inheritance. Default value: enabled.
Record LoadRunner Callback	Records the LoadRunner stub object as a callback. If disabled, VuGen records the original class as the callback. Default value: enabled.
Recorded Protocol	Specifies which protocol to record: RMI, CORBA, JMS, or Jacada. Default value: RMI.

UI Element (A-Z)	Description
Remove Lines Containing	Remove all lines containing one of the specified strings from the script. To specify multiple strings, separate the entries with commas. This feature is useful for customizing the script for a specific testing goal.
Unreadable Strings as Bytes	Represents strings containing unreadable characters as byte arrays. This option applies to strings that are passed as parameters to invocations. Default value: enabled.
Use _JAVA_OPTIONS flag	Forces JVM versions 1.2 and higher to use the _JAVA_OPTION environment variable which contains the desired JVM parameters. Default value: disabled.
Use DLL hooking to attach LoadRunner support	Use DLL hooking to automatically attach LoadRunner support to any JVM.

Recording Properties Serialization Options Node

Enables you to control how objects are serialized. Serialization is often relevant to displaying objects in an ASCII representation in order to parameterize their values.

To Access	Tools > Recording Options > Recording Properties > Serialization Options
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
See Also	"Correlating Java Scripts - Serialization" on page 192

User interface elements are described below:

UI Element (A-Z)	Description
Unfold Serialized Objects	<p>Expands serialized objects in ASCII representation and allows you to view the ASCII values of the objects in order to perform parameterization.</p> <ul style="list-style-type: none"> ▶ Limit Object Size (bytes). Limits serializable objects to the specified value. Objects whose size exceeds this value, will not be given ASCII representation in the script. Default value: 3072 bytes. ▶ Ignore Serialized Objects. Lists the serialized objects not to be unfolded when encountered in the recorded script. Separate objects with commas. Syntax: java.lang.Object class format, delimiter = "," ▶ Serialization Delimiter. Indicates the delimiter separating the elements in the ASCII representation of objects. VuGen will only parameterize strings contained within these delimiters. The default delimiter is '#'. ▶ Unfold Arrays. Expands array elements of serialized objects in ASCII representation. If you disable this option and an object contains an array, the object will not be expanded. Default value: enabled—all deserialized objects are totally unfolded. ▶ Limit Array Entries. Instructs the recorder not to open arrays with more than the specified number of elements. Default value: 200.

RTE Configuration Node

Enables you to set the recording options to match the character set used during terminal emulation.

To Access	Tools > Recording Options > RTE > Configuration
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Character Set	Match the character set used during terminal emulation. The default character set is ANSI. For Kanji and other multi-byte platforms, you can specify DBCS (Double-byte Character Set).

RTE Node

Enables you to set the general RTE recording options.

To Access	Tools > Recording Options > RTE > RTE
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Generate automatic synchronization commands	<p>Automatically generates a number of TE-synchronization functions, and insert them into the script while you record.</p> <ul style="list-style-type: none"> ▶ Cursor. Generate a TE_wait_cursor function before each TE_type function. ▶ Prompt. Generate a TE_wait_text function before each TE_type function (where appropriate). ▶ X-System. Generate a TE_wait_sync function each time a new screen is displayed while recording. <p>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.</p>
Generate automatic X-System transaction	<p>Records the time that the system was in the X SYSTEM mode during a scenario run. This is accomplished by inserting a TE_wait_sync_transaction function after each TE_wait_sync function. Each TE_wait_sync_transaction function creates a transaction with the name default. Each TE_wait_sync_transaction function records the time that the system spent in the previous X SYSTEM state.</p>
Generate screen header comments	<p>Generates screen header comments while recording a Vuser script, and inserts the comments into the script. A generated comment contains the text that appears on the first line of the terminal emulator window.</p> <p>Note: You can generate comments automatically only when using block-mode terminal emulators such as the IBM 5250.</p>
Keyboard record timeout	<p>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.</p>

SAPGUI Auto Logon Node

Enables you to log on automatically when you begin recording. The logon functions are placed in the vuser_init section of the script.

To Access	Tools > Recording Options > SAPGUI > Auto Logon
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Enable Auto logon	Enables you to log on automatically when you begin recording. Enter the Server name , User , Password , Client name, and interface Languages for the SAP server.

SAPGUI Code Generation Node

Enables you to set the code generation settings for the SAPGUI protocol.

To Access	Tools > Recording Options > SAPGUI > Code Generation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
Always generate Object ID in header file	Places the Object IDs in a separate header file instead of in the script. When you disable this option, VuGen generates the IDs according to the specified string length in the general script setting. This results in a more compact and cleaner script.
Generate Fill Data steps	Generates Fill Data steps for table and grid controls—instead of separate steps for each cell.
Generate logon operation as a single step	Generates a single <code>sapgui_logon</code> method for all of the logon operations. This helps simplify the code. If you encounter login problems, disable this option.

SAPGUI General Node

Enables you to set the general recording options for the SAPGUI protocol.

To Access	Tools > Recording Options > SAPGUI > General
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.

User interface elements are described below:

UI Element (A-Z)	Description
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.
Changing events during recording	<ul style="list-style-type: none"> ▶ Process Context menus by text. Processes context menus by their text, generating <code>sapgui_toolbar_select_context_menu_item_by_text</code> functions. When disabled, VuGen processes context menus by their IDs, and generates a <code>sapgui_toolbar_select_context_menu_item</code> for context menus. This is an advantage when working with Japanese characters.

Silverlight Services Node

Enables you to manage WSDL files in Silverlight Protocol scripts.

To Access	Tools > Recording Options > Silverlight > Services
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
Relevant Tasks	"How to Import WSDL Files" on page 873

User interface elements are described below:

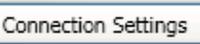
UI Element (A-Z)	Description
<Service list>	The list of imported WSDL files and their locations. The toolbar allows you to add, delete and edit WSDL files. Additionally, select the Protocol and Security Data button to edit Protocol and Security Data.
Automatically detect WSDL files and import services during code generation	Automatically attempts to locate and import WSDL files used in your script.
Do not use WSDL files	Disables WSDL files in your script, generating SOAP requests instead. This results in a lower level script, however it generally increases the script performance.
Service Endpoint	The location of the endpoint at which a given WSDL is available.
Use WSDL files included in the script	Enables WSDL files imported automatically or manually.
WSDL Location	The location of the selected WSDL file.

Add / Edit Services Dialog Box

Enables you to locate and import WSDLs to a Silverlight Protocol script.

To Access	Tools > Recording Options > Silverlight > Services > Add
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319.
Relevant Tasks	"How to Import WSDL Files" on page 873

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the Connection Settings dialog box, enabling you to configure the proxy and authentication information for the specified WSDL file. For user interface information, see "Connection Settings Dialog Box" on page 411.
Select WSDL from	<ul style="list-style-type: none"> ▶ URL. Select the WSDL by specifying the URL. ▶ File. Select the WSDL by specifying the local path. ▶ Previously Imported. Select the WSDL from the WSDL History (list of previously imported WSDL files).
Service Endpoint	The location of the endpoint at which a given WSDL is available.
WSDL Location	The URL or local path to the WSDL.

Connection Settings Dialog Box

Configures the proxy and authentication information for WSDL files in Silverlight Protocol scripts.

To Access	Tools > Recording Options > Silverlight > Services > Add > Connection Settings
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ▶ These settings are only relevant for importing WSDL files. To configure authentication and proxy information for WSDL files to be used during replay, add a web_set_user_step function with the desired values.
Relevant Tasks	"How to Import WSDL Files" on page 873

User interface elements are described below:

UI Element (A-Z)	Description
Authentication	Enable the authentication settings by selecting Use Authentication Settings and entering your user name and password.
Proxy	Enable the proxy settings by selecting Use Proxy Settings and entering your user name, password, server, and port number.

Protocol and Security Scenario Data Dialog Box

Enables you to configure the protocol and security scenario data settings.

To Access	Tools > Recording Options > Silverlight > Services > Protocol and Security Data Button
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 319. ▶ These settings are only relevant for importing WSDL files. To configure authentication and proxy information for WSDL files to be used during replay, add a web_set_user_step function with the desired values. ▶ The settings in this dialog box are reset during code generation.
Relevant Tasks	"How to Import WSDL Files" on page 873

User interface elements are described below:

UI Element (A-Z)	Description
Port	An individual endpoint for a WSDL binding. Note: Selecting a different port resets the values in this dialog box to the last saved values. Any changes made that have not yet been saved will be reset.
Description	A description of the selected security scenario.
Transport	The transport layer protocol used by VuGen to send service requests to the server. You can select HTTP, HTTPS, or LrHTTP. Note: HTTP is not compatible with UserNameOverTransport security and HTTPS requires that you select UserNameOverTransport security.
Encoding	The encoding method to be used for service requests sent to the server.
WS Addressing	The WS-Addressing version for the selected WSDL file.
Security	If the server requires authentication, select UserNameOverTransport as your authentication mode and enter a valid user name and password.

WinSocket Node

Enables you to set the WinSocket recording options.

To Access	Tools > Recording Options > WinSocket
Important Information	When using translation tables on Solaris machines, you must set the following environment variables on all machines running scripts: <pre>setenv LRSDRV_SERVER_FORMAT 0025 setenv LRSDRV_CLIENT_FORMAT 04e4</pre>
Relevant Tasks	"How to Record a WinSocket Script" on page 1174

User interface elements are described below:

UI Element (A-Z)	Description
	Adds a new entry to the list of excluded sockets.
	Removes the selected entry from the list of excluded sockets.
Do not include excluded socket in log	Excludes the sockets on the list from the log. Clearing this option enables logging for the excluded sockets. Their actions are preceded by "Exclude" in the log file.
Exclude Settings/ Socket List	<p>The host and port of the sockets to exclude from the recording or regeneration of the script. Use the following syntax:</p> <ul style="list-style-type: none"> ➤ host:port format excludes a specific port. ➤ host format excludes all ports for the specified host. ➤ :port format excludes a specific port on the local host. ➤ *:port format excludes a specific port on all hosts.

UI Element (A-Z)	Description
Think Time Threshold	<p>During recording, VuGen automatically inserts the think time steps when you pause between actions. You can set a threshold level, below which the recorded think time will be ignored.</p> <p>Default value: five seconds.</p>
Translation tables	<p>The Translation Table lets you specify the format for recording sessions when using the WinSock single protocol, and for code generation when using a WinSock multi protocol. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. Select a translation option from the list box.</p> <p>The first four digits of the listbox item represent the server format. The last four digits represent the client format.</p> <p>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.</p> <p>Note: If your data is in ASCII format, it does not require translation. Select the None option, the default value.</p>

11

Run-Time Settings

This chapter includes:

Concepts

- ▶ Run-Time Settings Overview on page 418
- ▶ Error Handling on page 418
- ▶ Content Checking Overview on page 419
- ▶ Logging CtLib Server Messages on page 420
- ▶ Multithreading on page 421

Reference

- ▶ Protocol Compatibility Table on page 422
- ▶ Run-Time Settings User Interface on page 428

Concepts

Run-Time Settings Overview

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.

Error Handling

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

- ▶ Using run-time settings. You can specify the **Continue on Error** run-time setting. The **Continue on Error** run-time setting applies to the entire Vuser script. You can use the `lr_continue_on_error` function to override the **Continue on Error** run-time setting for a portion of a script. For details, see "Error Handling" on page 418.
- ▶ 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);
```

Content Checking Overview

You use the ContentCheck run-time options to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text **ASP Error**. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay. Note that VuGen searches the body of the pages—not the headers.

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.

Multithreading

The Controller uses a driver program (such as *mdrv.exe* or *r3vuser.exe*) to run your Vusers. If you run each Vuser as a process, then the same driver program is launched (and loaded) into the memory again and again for every instance of the Vuser. Loading the same driver program into memory uses up large amounts of RAM (random access memory) and other system resources. This limits the numbers of Vusers that can be run on any load generator.

Alternatively, if you run each Vuser as a thread, the Controller launches only one instance of the driver program (such as *mdrv.exe*), for every 50 Vusers (by default). This driver process/program launches several Vusers, each Vuser running as a thread. These threaded Vusers share segments of the memory of the parent driver process. This eliminates the need for multiple re-loading of the driver program/process saves much memory space, thereby enabling more Vusers to be run on a single load generator.

To configure these options, see "General Miscellaneous Node" on page 449.

Reference

Protocol Compatibility Table

The following table lists the types of Vuser scripts and which run-time setting nodes are available for each type.

Protocol	Run-Time Setting Nodes
AMF	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, Content Check
AJAX	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
Ajax TruClient	<ul style="list-style-type: none"> ▶ General - Pacing, Additional Attributes, Log, Other Settings
C Vuser	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Citrix	<ul style="list-style-type: none"> ▶ General – Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network – Speed Simulation ▶ Citrix – Configuration, Synchronization
COM/DCOM	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
DB2 CLI	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous

Protocol	Run-Time Setting Nodes
DNS	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
EJB	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Java Environment Settings - Classpath, Java VM
FTP	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation
Flex	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, Content Check
i-mode	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters
Informix	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
IMAP	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, Content Check
Java over HTTP	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Java Environment Settings - Classpath, Java VM

Protocol	Run-Time Setting Nodes
Java Record Replay, Java Vuser	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Java Environment Settings - Classpath, Java VM
Javascript Vuser	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Internet Protocol - Proxy, Preferences, Download Filters
LDAP	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation
MMS (Media Player)	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation
.NET	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ .NET - .NET
RDP	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ RDP - Configuration, Synchronization, Advanced, RDP Agent
MAPI	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
MS SQL Server	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
MMS (Multimedia Messaging Service)	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Browser - Browser Emulation ▶ MMS - Server and Protocol ▶ WAP - Radius, Gateway ▶ Internet Protocol - Proxy, Preferences, Download Filters

Protocol	Run-Time Setting Nodes
ODBC	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Oracle (2-Tier)	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Oracle NCA	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Oracle NCA - Client Emulation
Oracle Web Applications 11i	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck ▶ Oracle NCA - Client Emulation
PeopleSoft Enterprise	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
PeopleSoft Tuxedo	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
POP3	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation
Real	<ul style="list-style-type: none"> ▶ General -Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation

Protocol	Run-Time Setting Nodes
SAP Web	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SAP Click and Script	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SAPGUI	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ SAPGUI - General
Siebel Web	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
Silverlight	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Silverlight - Services ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SMTP	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation

Protocol	Run-Time Setting Nodes
Sybase CTlib / DBlib	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
RTE	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ RTE - RTE
Tuxedo, Tuxedo 6	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
VB Script Vuser	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Internet Protocol - Proxy, Preferences, Download Filters
VB Vuser	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Internet Protocol - Proxy, Preferences, Download Filters ▶ VBA - VBA
WAP	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ WAP - Radius, Gateway ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
Web (Click and Script)	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck

Protocol	Run-Time Setting Nodes
Web (HTTP/HTML)	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ Browser - Browser Emulation ▶ Internet Protocol - Proxy, Preferences, Download Filters, Content Check ▶ Data Format Extension - Configuration
Web Services	<ul style="list-style-type: none"> ▶ General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation ▶ JMS - Advanced
Windows Sockets	<ul style="list-style-type: none"> ▶ General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous ▶ Network - Speed Simulation

Run-Time Settings User Interface

This section includes:

- ▶ Browser Emulation Node on page 430
- ▶ Citrix Configuration Node on page 434
- ▶ Citrix Synchronization Node on page 435
- ▶ Flex Externalizable Node on page 436
- ▶ Flex RTMP Node on page 437
- ▶ General Additional Attributes Node on page 437
- ▶ General Log Node on page 438
- ▶ General Miscellaneous Node on page 449
- ▶ General Pacing Node on page 451
- ▶ General Run Logic Node on page 451
- ▶ General Think Time Node on page 452

- Internet Protocol ContentCheck Node on page 454
- Internet Protocol Download Filters Node on page 455
- Internet Protocol Preferences Node on page 456
- Internet Protocol Proxy Node on page 471
- Java Classpath Node on page 473
- Java VM Node on page 474
- JMS Advanced Node on page 474
- Microsoft .NET Environment Node on page 477
- MMS Server and Protocol Node on page 478
- Network Speed Simulation Node on page 479
- Oracle NCA Client Emulation Node on page 480
- RDP Advanced Node on page 482
- RDP Agent Node on page 484
- RDP Configuration Node on page 485
- RDP Synchronization Node on page 486
- RTE Node on page 487
- SAPGUI General Node on page 489
- VBA Node on page 491
- WAP Gateway Node on page 492
- WAP Radius Node on page 496

Browser Emulation Node

Enables you to configure the browser related run-time settings.

To Access	Vuser > Run-Time Settings> Browser > Browser Emulation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
User Agent	<p>This displays information about the browser that is to be emulated.</p> <p>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.</p> <p>Click  to change the User Agent header. Specify the browser type, browser version, language, and operating system or enter a custom User Agent header.</p>
Change Button	Opens the User-Agent Dialogue Box, enabling you to change the emulated browser.

UI Element (A-Z)	Description
Simulate Browser Cache	<p>Instructs the Vuser to simulate a browser with a cache (enabled by default). Even if you disable this option, each resource is only downloaded once for each page, even if it appears multiple times. A resource can be an image, a frame, or another type of script file. Enabling this options allows you to set the following options:</p> <ul style="list-style-type: none"> ▶ Cache URLs requiring content (HTML). Instructs VuGen to cache only the URLs that require the HTML content. The content may be necessary for parsing, verification, or correlation. When you select this option, HTML content is automatically cached. Default value: enabled. Tip: To decrease the memory footprint of the virtual users, disable this option, unless it is an explicit requirement for your test. ▶ Check for newer versions of stored pages every visit to the page. Instructs the browser to check for later versions of the specified URL than those stored in the cache. When you enable this option, VuGen adds the "If-modified-since" attribute to the HTTP header. This option brings up the most recent version of the page, but also generates more traffic during the scenario or session execution. Default value: disabled. ▶ Advanced... . Opens the Cache URL Requiring Content - Advanced Dialog Box, enabling you to specify the URL content types that you want to store in the cache.

UI Element (A-Z)	Description
Download non-HTML resources	<p>Instructs Vusers to load graphic images when accessing a Web page during replay. This includes both graphic images that were recorded with the page, and those which were not explicitly recorded along with the page. When real users access a Web page, they wait for the images to load. Therefore, enable this option if you are trying to test the entire system, including end-user time. To increase performance and not emulate real users, disable this option.</p> <p>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).</p>
Simulate a new user each iteration	<p>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.</p> <p>Default value: enabled.</p> <p>► 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.</p>

Cache URL Requiring Content - Advanced Dialog Box

The Advanced dialog box lets you specify the URL content types that you want to store in the cache.

User interface elements are described below:

UI Element (A-Z)	Description
	Restore the default content types if they were deleted.
	Add. Adds a new content type.
	Remove. Removes a content type.
<Content Types List>	A list of the content types along with a check box to enable/disable each type.
Specify URLs requiring content in addition to HTML page	Enables you to specify URL's

Citrix Configuration Node

Enables you to set the Citrix configuration run-time settings.

To Access	Vuser > Run-Time Settings > Citrix > Configuration
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Queue mouse movements and keystrokes	Instructs Vusers to create a queue of mouse movements and keystrokes, and send them as packets to the server less frequently. This setting reduces network traffic with slow connections. Enabling this option makes the session less responsive to keyboard and mouse movements. Default value: disabled.
Sound quality	Specifies the quality of the sound. If the client machine does not have a 16-bit Sound Blaster-compatible sound card, select Sound Off . With sound support enabled, you will be able to play sound files from published applications on your client machine.
SpeedScreen Latency Reduction	The mechanism used to enhance user interaction when the network speed is slow. You can turn this mechanism on or off , depending on the network speed. The auto option turns it on or off based on the current network speed. If you do not know the network speed, set this option to Use Server Default to use the machine's default.

UI Element (A-Z)	Description
Use data compression	Instructs Vusers to compress the transferred data. You should enable data compression if you have a limited bandwidth. Default value: enabled.
Use disk cache for bitmaps	Instructs Vusers to use a local cache to store bitmaps and commonly-used graphical objects. You should enable this option if you have a limited bandwidth. Default value: disabled.

Citrix Synchronization Node

Enables you to set the Citrix synchronization run-time settings.

To Access	Vuser > Run-Time Settings > Citrix > Synchronization
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Default Image Sync Tolerance	This setting controls the level of equality two images must share to be considered synchronized. Exact. Must have a 100% match. Low. Must have a 95 % match. Medium. Must have an 85% match. High. Must have a 70% match. Default value: Exact.

UI Element (A-Z)	Description
Timeout	<ul style="list-style-type: none"> ▶ Connect Time. The time (in seconds) to wait idly at an established connection before exiting. ▶ Waiting Time. The time (in seconds) to wait idly at a synchronization point before exiting.
Typing rate	The delay (in milliseconds) between keystrokes.

Flex Externalizable Node

Enables you to set the Flex externalizable run-time settings.

To Access	Vuser > Run-Time Settings > Flex > Externalizable
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Encoded AMF3 using external parser	This is automatically checked if you recorded externalizable objects using an external parser. Modifying this option is not recommended.
Encoded externalizable objects with LoadRunner parser	This is automatically checked if you recorded externalizable objects using the LoadRunner parser. Modifying this option is not recommended.
Flex server/application JAR files location	This allows you to edit/update the location of the JAR files if they have changed after recording.

Flex RTMP Node

Enables you to set the Flex RTMP run-time settings.

To Access	Vuser > Run-Time Settings > Flex > RTMP
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Close all open RTMP connections after each iteration	Automatically disconnects any open RTMP connections at the end of each iteration.

General Additional Attributes Node

You can use the Additional Attributes node to provide additional arguments for a Vuser script. The Additional Attributes settings apply to all Vuser script types.

You specify command line arguments that you can retrieve at a later point during the test run, using `lr_get_attrib_string`. Using this node, you can pass external parameters to prepared scripts.

To Access	Vuser > Run-Time Settings > General > Additional Attributes
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
<Additional Attributes table>	A list of the additional attributes and their values.
	Add a new argument.
	Remove an argument.

General Log Node

Enables you to configure the amount and types of information that is recorded in the log.

To Access	Vuser > Run-Time Settings > General > Log
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422. ▶ You can also manually program a Vuser script to send messages to an output log by using the lr_error_message and lr_output_message functions. ▶ For Web protocols, you can obtain more information in the log by editing 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.

User interface elements are described below:

UI Element (A-Z)	Description
Enable logging	Enables automatic logging during replay. Disabling this options only affects automatic logging and log messages issued through Ir_log_message . Messages sent manually, using Ir_message , Ir_output_message , and Ir_error_message , are still issued.
Log options	Indicates when messages are sent to the log: <ul style="list-style-type: none"> ▶ Send messages only when an error occurs. Sends messages to the log only when an error occurs. Click Advanced to configure the log cache size. If the contents of the cache exceed the specified size, VuGen deletes the oldest items. ▶ Always send messages. Sends all messages to the log.
Standard log	Creates a standard log of functions and messages sent during script execution to use for debugging. You can disable this option for large load testing scenarios or profiles if you wish to save system resources.
Extended log	Creates an extended log, including warnings and other messages. You can disable this option for large load testing scenarios or profiles if you wish to save system resources. You can also specify the following options: <ul style="list-style-type: none"> ▶ Parameter substitution. Logs all parameters assigned to the script along with their values. ▶ Data returned by server. Logs all of the data returned by the server. ▶ Advanced trace. Logs all of the functions and messages sent by the Vuser during the session.

General Log Node - TruClient

Enables you to configure the amount and types of information that is reported to a log for TruClient scripts.

To Access	Vuser > Run-Time Settings > General > Log
-----------	---

User interface elements are described below:

UI Element (A-Z)	Description
Log Level and Associated Options	<p>Error-only</p> <ul style="list-style-type: none"> ▶ Select to log only error messages. <p>Standard log</p> <ul style="list-style-type: none"> ▶ Log errors/warnings only (in Standard log). Select to log errors and warnings only. Informative messages are not included. <p>Note: In Extended log, all messages ignored due to this option are included, regardless of this setting.</p> <p>Extended log</p> <ul style="list-style-type: none"> ▶ Select to log low-level informative messages and all messages included in the Standard log. ▶ HTTP-related messages <ul style="list-style-type: none"> ▶ Log HTTP request headers. Log HTTP request header of each request. ▶ Log HTTP request body (POST data). Log the HTTP request body of each request. ▶ Log HTTP response headers. Log the HTTP response headers of each response. ▶ Log HTTP response body. Log the HTTP response body of each response. ▶ Log HTTP trace. Log the HTTP request/response handling ▶ Application under Test (AUT) messages <ul style="list-style-type: none"> ▶ Log AUT error messages. Logs error messages which are received from the application under test. ▶ Log AUT non-error messages. Log informative messages which are received from the application under test. ▶ Parameterization <ul style="list-style-type: none"> ▶ Log Parameter substitution. Logs the parameters and the values used when the Vuser script runs.

UI Element (A-Z)	Description
Create a log file only if required	Accumulate messages in memory, write only if required. <ul style="list-style-type: none"> ▶ Select to log messages only when an error occurs. ▶ Accumulate Messages in Memory Buffer Size. Enter the memory buffer size where important messages are logged. <p>Note: If the size is too small, messages will be lost. If the size is too large, this may contribute to paging issues and slow execution.</p>
Log Message Formatting Options	User Log Line Length. <ul style="list-style-type: none"> ▶ Enter the limit for a single line in a user log file before the message line begins to wrap. Include Timestamps in user log. <ul style="list-style-type: none"> ▶ Select to include time trace to each message in the log.
Internal Support Options	These options are for use with a customer support representative only. Do not modify them unless specifically instructed to do so by customer support.

General Other Settings Node

Enables you to configure the amount and types of information that is recorded in the log for TruClient scripts.

To Access	Vuser > Run-Time Settings > General > Other Settings
-----------	--

User interface elements are described below:

UI Element (A-Z)	Description
Simulate a new user on each iteration	<p>Instructs VuGen to reset all HTTP contexts between iterations. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session.</p> <p>Default value: enabled.</p>
Proxy selection	<p>The settings in this section apply to both recording and running the script. They apply to this script only. To configure proxy settings for all TruClient scripts, open the Browser Configuration settings dialog box. For more information, see "Configure the Global Browser Configuration Settings" on page 512.</p> <p>Retrieve global proxy settings (defined in Ajax TruClient Browser Options).</p> <ul style="list-style-type: none"> ▶ Select to use the proxy settings defined in the Browser Options dialog box. The settings are retrieved when you click Develop Script. Once global proxy settings are retrieved, they are copied to the script's run-time settings and can be managed in the "Define proxy settings for script" section. <p>Define specific proxy settings for the script</p> <ul style="list-style-type: none"> ▶ No proxy (direct connection to the internet). For all Vusers, make the connection to the Internet directly without using a proxy server. ▶ Manual proxy configuration. If selected, you can specify exceptions to the proxy server rules, specify the proxy server for all HTTP/HTTPS connections. you require a separate proxy server for HTTP connections, check Use separate proxy for HTTPS protocol box and specify the proxy server settings. ▶ AUTOMATIC proxy configuration (PAC). If selected, the proxy settings is automatically read from the proxy auto configuration file (PAC). In the URL field, enter the URL of the PAC file.

UI Element (A-Z)	Description
<p>Snapshots generation</p>	<p>Recording snapshots generation</p> <ul style="list-style-type: none"> ▶ Never. If selected, recording snapshots of the script are not automatically saved. ▶ Always. If selected, recording snapshots are automatically saved for every step in the script. <p>Replay snapshots generation</p> <ul style="list-style-type: none"> ▶ Never. If selected replay snapshots are not generated. <p>Note: This is the default setting.</p> <ul style="list-style-type: none"> ▶ On error. Select to generate a snapshot when errors occurs. This option can be used to help identify bugs in your script. For more information, see "Debug Scripts Using Snapshots" on page 518. ▶ Always. If selected snapshots are generated for every step in the script.
<p>Action on error</p>	<p>Abort script</p> <ul style="list-style-type: none"> ▶ Select to abort running script when an error occurs. <p>Continue to the next iteration</p> <ul style="list-style-type: none"> ▶ Select to stop iteration when an error occurs and continue to the next one.

UI Element (A-Z)	Description
Replay options	<p>Maximum time for object-not-found (seconds)</p> <ul style="list-style-type: none">▶ Enter the maximum time to search for an object before the application displays an error message. <p>Inter-step interval (milliseconds)</p> <ul style="list-style-type: none">▶ Enter the minimal interval between steps. Specifying a higher value can help with synchronization issues, but too high a value may slow the script more than necessary. <p>End-of-network identification timeout (milliseconds)</p> <ul style="list-style-type: none">▶ The end-of-network for a step is recognized when the specified time has elapsed with no networking activity.

UI Element (A-Z)	Description
Advanced	<p>A number of options for advanced users. For more details, read the tips displayed at the bottom of the Run-Time settings dialog box.</p> <ul style="list-style-type: none"> ➤ Home Page URL. Enter the URL to navigate to when window home is called from JavaScript. ➤ User-Agent. Enter the User-Agent string for overriding the browser default in request headers. ➤ Compare the page in cache to the page on the network. Enter the corresponding number used to compare the URL page in the cache to the URL page called from the network. 0=Once per session, 1=Every time the page is accessed, 2=Never, 3=When the page is out of date (default). ➤ Keep-Alive. Select to allow persistent (non-proxied) network connections. Enter the following corresponding number: 1=Allow persistent (non-proxied) network connections, so that the open connections can be reused, 0=Close each connection after the request is complete. ➤ Proxy Keep-Alive. Select to allow persistent proxy connections. Enter the following corresponding number: 1=Allow persistent proxy connections, so that open connections can be reused. 0=Close each connection after the request is complete. ➤ Keep-Alive timeout (sec). Enter the number of seconds to keep idle connections open. ➤ HTTP version. Enter the HTTP version to use when accessing the network/application not via a proxy.. ➤ Proxy HTTP version. Enter the Proxy HTTP version to use when accessing the network/application via a proxy. ➤ DNS cache entries. Enter the max number of entries to keep in the DNS cache. Note: Enter 0 to disable this option. ➤ Non-interactive window size. Enter the initial dimensions (width and height in pixels) of the browser window in non-interactive mode. ➤ SSL. Select the secure connection setting/s <ul style="list-style-type: none"> ➤ SSL 2.0 ➤ SSL 3.0 ➤ TLS 1.0

General Miscellaneous Node

Enables you set miscellaneous run-time settings.

To Access	Vuser > Run-Time Settings > General > Miscellaneous
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422. ▶ We do not recommend enabling both the Continue on Error and Generate Snapshot on Error options in a load test environment. This configuration may adversely affect the Vusers' performance. ▶ The following protocols should not be run as threads: Sybase-Ctlib, Sybase-Dblib, Informix, Tuxedo, and PeopleSoft-Tuxedo. ▶ If you require the Vusers to generate breakdown data for diagnostics (J2EE) during the scenario run, do not use automatic transactions. Instead, manually define the beginning and end of each transaction.

User interface elements are described below:

UI Element (A-Z)	Description
<p>Automatic Transactions</p>	<ul style="list-style-type: none"> ▶ Define each action as a transaction. Instructs LoadRunner (not applicable to HP Business Service Management) to handle every action in the script as a transaction. ▶ Define each step as a transaction. Instructs LoadRunner (not applicable to HP Business Service Management) to handle every step in the script as a transaction.
<p>Error Handling</p>	<ul style="list-style-type: none"> ▶ Continue on Error. Continue script execution when an error occurs. Default value: disabled. ▶ Fail open transactions on lr_error_message. Instructs VuGen to mark all transactions in which an lr_error_message function was issued, as <i>Failed</i>. The lr_error_message function is issued through a manually defined <i>If</i> statement. ▶ Generate Snapshot on Error. Generates a snapshot when an error occurs. You can see the snapshot by viewing the Vuser Log and double-clicking on the line at which the error occurred. <p>For concept details, see (XXX ADD X-REFERENCE).</p>
<p>Multithreading</p>	<ul style="list-style-type: none"> ▶ Run Vuser as a process. Disables multithreading. ▶ Run Vuser as a thread. Enables multithreading.

General Pacing Node

Allows you control the time between iterations. The pace tells the Vuser how long to wait between iterations of your actions.

To Access	Vuser > Run-Time Settings > General > Pacing
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
After the previous iteration ends:	Starts each new iteration a specified amount of time after the end of the previous iteration. Specify either an exact number of seconds or a range of time.
As soon as the previous iteration ends	The new iteration begins as soon as possible after the previous iteration ends.
At <fixed/random> intervals	You specify the time between iteration—either a fixed number of seconds or a range of seconds from the beginning of the previous iteration. Each scheduled iteration will only begin when the previous iteration is complete.

General Run Logic Node

Enables you to set the run logic run-time settings.

To Access	Vuser > Run-Time Settings > General > Run Logic
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.
See Also	"Script Sections" on page 34

User interface elements are described below:

UI Element (A-Z)	Description
	Inserts a new Action at the insertion point.
	Inserts a new Action block at the insertion point.
	Deletes an item.
	Moves an item up.
	Moves an item down.
	Opens the Properties Dialog Box enabling you to set the run logic and iterations settings. Run Logic. Configures the action to run sequentially or randomly. Iterations. Sets the number of times an item will run.
<Run logic tree>	A graphical illustration of the run logic for this script.
Number of Iterations	The number of times the script will run of the items in the run logic tree.

General Think Time Node

Enables you to configure the think time settings, controlling the time that a VuGen waits between actions. These settings are designed to help you emulate a real user.

To Access	Vuser > Run-Time Settings > General > Think Time
------------------	--

Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.
See Also	► VuGen uses lr_think_time functions to record think time values into your Vuser scripts. For more information about the lr_think_time function and how to modify it manually, see the <i>Online Function Reference</i> (Help > Function Reference).

User interface elements are described below:

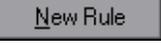
UI Element (A-Z)	Description
As recorded	During replay, use the argument that appears in the lr_think_time function. For example, lr_think_time(10) waits ten seconds.
Ignore think time	Ignore the recorded think time—replay the script ignoring all lr_think_time functions.
Limit think time to	Limit the think time's maximum value.
Multiply recorded think time by	During replay, use a multiple of the recorded think time. This can increase or decrease the think time applied during playback. For example, if a think time of four seconds was recorded, you can instruct your Vuser to multiply that value by two, for a total of eight seconds. To reduce the think time to two seconds, multiply the recorded time by 0.5.
Replay the think time	Enables options which let you customize the recorded think times.
Use random percentage of recorded think time	Use a random percentage of the recorded think time. You set a range for the think time value by specifying a range for the think time. For example, if the think time argument is 4, and you specify a minimum of 50% and a maximum of 150%, the lowest think time can be two (50%) and the highest value six (150%).

Internet Protocol ContentCheck Node

Enables you to check websites for content during run-time.

To Access	Vuser > Run-Time Settings > Internet Protocol > ContentCheck
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
	Deletes the selected rule or application.
	Exports rules to an xml file.
	Imports rules from an existing xml file.
	Adds a new application to the list of applications and rules. Click to change the name.
	Displays the rule criteria in the right pane, allowing you to enter a new rule for the currently selected application. <ul style="list-style-type: none"> ➤ Search for text. The text of the string for which you want to search. ➤ Search by prefix and suffix. The prefix and suffix of the string for which you want to search. ➤ Match case. Perform a case sensitive search. ➤ Fail if. Configure the results of the search to fail if the string is either found or not found. ➤ Search JavaScript alert box text. Only search for text within JavaScript alert boxes (Web (Click and Script), PeopleSoft Enterprise, and Oracle Web Applications 11i Users only).
	Set the ContentCheck configuration as default (i.e. new scripts will start with this configuration).

UI Element (A-Z)	Description
<Application and Rule list>	A list of applications and rules. You can enable and disable individual items by using the check boxes to the left of each item.
Enable ContentCheck during replay	Enable content checking during replay. Note that even after you define applications, you can disable it for a specific test run, by disabling this option. Default value: enabled.

Internet Protocol Download Filters Node

Enables you to set the download filters.

To Access	Vuser > Run-Time Settings > Internet Protocol > Download Filters
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
<Filter list>	A list of filters for the script. Each filter has a type and data. For example, a filter of type URL would have a URL address as its data. You can Add , Edit , Remove , or Remove All entries in the list.
Exclude addresses in list	Ignore requests from the listed sites or hosts.
Include only addresses in list	Restrict replay to the listed sites or hosts.

Internet Protocol Preferences Node

Enables you to set various internet related run-time settings.

To Access	Vuser > Run-Time Settings > Internet Protocol > Preferences
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Checks	<p>► Enable Image and Text Check. Allows the Vuser to perform verification checks during replay by executing the verification functions web_find or web_image_check. This option only applies to statements recorded in HTML-based mode. Vusers running with verification checks use more memory than Vusers who do not perform checks.</p> <p>Default value: disabled.</p>

UI Element (A-Z)	Description
Generate Web Performance Graphs	<p>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.</p> <p>Note: If you do not use the Web performance graphs, disable these options to save memory.</p>

UI Element (A-Z)	Description
Advanced	<ul style="list-style-type: none"> <li data-bbox="586 222 1215 765"> <p>➤ WinInet Replay. Instructs VuGen to use the WinInet replay engine instead of the standard Sockets replay. VuGen has two HTTP replay engines: Sockets-based (default) or WinInet based. The WinInet is the engine used by Internet Explorer and it supports all of the features incorporated into the IE browser. The limitations of the WinInet replay engine are that it is not scalable and does not support 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.</p> <p>Default value: disabled (socket-based replay engine).</p> <li data-bbox="586 817 1205 979"> <p>➤ 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.</p> <p>Default value: enabled</p> <li data-bbox="586 996 1215 1308"> <p>➤ Non-critical item errors as warnings. This option returns a warning status for a function which failed on an item that is not critical for load testing, such as an image or Java applet that failed to download. This option is enabled by default. If you want a certain warning to be considered an error and fail your test, you can disable this option. You can set a content-type to be critical by adding it to the list of Non-Resources. For more information, see "Non-Resources Dialog Box" on page 367.</p> <li data-bbox="586 1326 1210 1447"> <p>➤ 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.</p> <li data-bbox="586 1465 1219 1482"> <p>➤ Options... . Opens the Advanced Options Dialog Box.</p>

Advanced Options Dialog Box

Enables you to set the advanced internet preference run-time settings.

To Access	Vuser > Run-Time Settings > Internet Protocol > Preferences > Options
Important Information	<ul style="list-style-type: none"> ▶ This dialog box is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422. ▶ This dialog box divides the properties into different categories: HTTP, General, Authentication, Log, and Web (Click and Script) Specific.

HTTP

UI Element (A-Z)	Description
HTTP version	<p>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.</p> <p>Default value: HTTP 1.1.</p> <p>HTTP 1.1 supports the following features:</p> <ul style="list-style-type: none"> ▶ Persistent Connections—see "Keep-Alive HTTP connections" below. ▶ HTML compression—see Accept Server-Side Compression below. ▶ Virtual Hosting—multiple domain names sharing the same IP address.
Keep-Alive HTTP connections	<p>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.</p> <p>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.</p> <p>Default value: enabled.</p>

UI Element (A-Z)	Description
Accept-Language request header	Provides a comma-separated list of accepted languages. For example, en-us, fr , and so forth. For more details, see "Page Request Header Language" on page 1220.
HTTP errors as warnings	Issues a warning instead of an error upon failing to download resources due to an HTTP error.
HTTP-request connect timeout (seconds)	The time, in seconds, that a Vuser will wait for the connection of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. Note that this timeout also applies to the time the Vuser will wait for a WAP connection, initiated by the wap_connect function. Default value: 120 seconds.
HTTP-request receive timeout (seconds)	The time, in seconds, that a Vuser will wait to receive the response of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. Default value: 120 seconds.

UI Element (A-Z)	Description
Request Zlib Headers	<p>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.</p> <p>Default value: Yes.</p>
Accept Server-Side Compression	<p>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.</p> <p>Default value: Accept gzip and deflate compression.</p> <p>To manually add compression, enter the following function at the beginning of the script:</p> <pre>web_add_auto_header("Accept-Encoding", "gzip");</pre> <p>To verify that the server sent compressed data, search for the string Content -Encoding: gzip in the section of the server's responses of the replay log. The log also shows the data size before and after decompression.</p>

General

UI Element (A-Z)	Description
DNS caching	<p>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.</p> <p>Default value: enabled.</p>
Convert from/to UTF-8	<p>Converts received HTML pages and submitted data from and to UTF-8. You enable UTF-8 support in the recording options. For more information, see "Recording Options" on page 307.</p> <p>Default value: no.</p>
Step timeout caused by resources is a warning	<p>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.</p> <p>Default value: disabled.</p>
Parse HTML Content-Type	<p>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.</p> <p>Default value: TEXT.</p> <p>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.</p>
Step download timeout (sec)	<p>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.</p>

UI Element (A-Z)	Description
Network buffer size	Sets the maximum size of the buffer used to receive the HTTP response. If the size of the data is larger than the specified size, the server will send the data in chunks, increasing the overhead of the system. When running multiple Vusers from the Controller, every Vuser uses its own network buffer. This setting is primarily for advanced users who have determined that the network buffer size may affect their script's performance. The default is 12K bytes. The maximum size is 0x7FFF FFFF.
Print NTLM information	Print information about the NTLM handshake to the standard log.
Print SSL information	Print information about the SSL handshake to the standard log.
Max number of error matches issued as ERRORS	Limits the number of error matches issued as ERRORS for content checks using a LB or RB (left boundary or right boundary). This applies to matches where a failure occurs when the string is found (Fail=Found). All subsequent matches are listed as informational messages. Default value: 10 matches.
Maximum number of META Refresh to the same page	The maximum number of times that a META refresh can be performed per page. Default value: 2.
ContentCheck values in UTF-8	Store the values in the ContentCheck XML file in UTF-8.

Authentication

UI Element (A-Z)	Description
Fixed think time upon authentication retry (msec)	Automatically adds a think time to the Vuser script for emulating a user entering authentication information (username and password). This think time will be included in the transaction time. Default value: 0.
Disable NTLM2 session security	Use full NTLM 2 handshake security instead of the more basic NTLM 2 session security response. Default value: No.
Use Windows native NTLM implementation	Use the Microsoft Security API for NTLM authentication instead of the indigenous one. Default value: No.
Enable integrated Authentication	Enable Kerberos-based authentication. When the server proposes authentication schemes, use Negotiate in preference to other schemes. Default value: No.
Induce heavy KDC load	Do not reuse credentials obtained in previous iterations. Enabling this setting will increase the load on the KDC (Key Distribution Server). To lower the load on the server, set this option to Yes in order to reuse the credentials obtained in previous iterations. This option is only relevant when Kerberos authentication is used. Default value: No.

Log

UI Element (A-Z)	Description
Print buffer line length	Line length for printing request/response header/body and/or JavaScript source, disabling wrapping.
Print buffer escape only binary zeros	<ul style="list-style-type: none"> ➤ 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

UI Element (A-Z)	Description
General	<ul style="list-style-type: none"> ▶ Home Page URL. The URL of the home page that opens with your browser (default is about:blank). ▶ DOM-based snapshots. Instructs VuGen to generate snapshots from the DOM instead of from the server responses. Default value: Yes. ▶ Charset conversions by HTTP. Perform charset conversions by the 'Content-Type:.....; charset=...' HTTP response header. Overrides 'Convert from /to UTF-8.' ▶ Reparsing when META changes charset. Reparse HTML when a META tag changes the charset. Effective only when Charset conversions by HTTP is enabled. Auto means reparsing is enabled only if it used in the first iteration. ▶ Fail on JavaScript error. Fails the Vuser when a JavaScript evaluation error occurs. Default value: No (issue a warning message only after a JavaScript error, but continue to run the script). ▶ Initialize standard classes for each new window project. When enabled, the script—the src compiled script, will not be cached. ▶ Ignore acted on element being disabled. Ignore the element acted on by a Vuser script function being disabled.

UI Element (A-Z)	Description
<p>Timers</p>	<ul style="list-style-type: none"> <p>➤ Optimize timers at end of step. When possible, executes a <code>setTimeout/setInterval/<META refresh></code> that expires at the end of the step before the expiration time Default value: Yes.</p> <p>➤ Single <code>setTimeout/setInterval</code> threshold (seconds). Specifies an upper timeout for the <code>window.setTimeout</code> and <code>window.setInterval</code> methods. If the delay exceeds this timeout, these methods will not invoke the functions that are passed to them. This emulates a user waiting a specified time before clicking on the next element. Default value: 5 seconds.</p> <p>➤ Accumulative <code>setTimeout/setInterval</code> threshold (seconds). Specifies a timeout for the <code>window.setTimeout</code> and <code>window.setInterval</code> methods. If the delay exceeds this timeout, additional calls to <code>window.setTimeout</code> and <code>window.setInterval</code> will be ignored. The timeout is accumulative per step. Default value: 30 seconds.</p> <p>➤ Reestablish <code>setInterval</code> at end of step. 0 = No; 1 = Once; 2 = Yes.</p>
<p>History</p>	<ul style="list-style-type: none"> <p>➤ History support. Enables support for the <code>window.history</code> object for the test run. The options are Enabled, Disabled, and Auto. The Auto option instructs Vusers to support the <code>window.history</code> object only if it was used in the first iteration. Note that by disabling this option, you improve performance. Default value: auto.</p> <p>➤ Maximum history size. The maximum number of steps to keep in the history list. Default value: 100 steps.</p>

UI Element (A-Z)	Description
Navigator Properties	<ul style="list-style-type: none"><li data-bbox="621 227 1252 387">▶ navigator.browserLanguage. The browser language set in the navigator DOM object's browserLanguage property. Default value: The recorded value. Scripts created with older recording engines use en-us by default.<li data-bbox="621 404 1252 564">▶ navigator.systemLanguage. The system language set in the navigator DOM object's systemLanguage property. Default value: The recorded value. Scripts created with older recording engines use en-us by default.<li data-bbox="621 581 1252 706">▶ navigator.userLanguage. The user language set in the navigator DOM object's userLanguage property. Default value: The recorded value. Scripts created with older recording engines use en-us by default.

UI Element (A-Z)	Description
<p>Screen Properties</p>	<ul style="list-style-type: none"> ▶ screen.width Sets the width property of the screen DOM object in pixels. Default value: 1024 pixels. ▶ screen.height Sets the height property of the screen DOM object in pixels. Default value: 768 pixels. ▶ screen.availWidth Sets the availWidth property of the screen DOM object in pixels. Default value: 1024 pixels. ▶ screen.availHeight. Sets the availHeight property of the screen DOM object in pixels. Default value: 768 pixels.
<p>Memory Management</p>	<ul style="list-style-type: none"> ▶ Default block size for DOM memory allocations. Sets the default block size for DOM memory allocations. If the value is too small, it may result in extra calls to malloc, slowing the execution times. Too large a block size, may result in an unnecessarily big footprint. Default value: 16384 bytes. ▶ Memory Manager for dynamically-created DOM objects. Yes—Use the Memory Manager for dynamically-created DOM objects. No—Do not use the Memory Manager, for example when multiple DOM objects are dynamically created in the same document as under SAP. Auto—Use the protocol recommended (default Yes for all protocols except for SAP). ▶ JavaScript Runtime memory size (KB). Specifies the size of the JavaScript runtime memory in kilobytes. Default value: 256 KB. ▶ JavaScript Stack memory size (KB). Specifies the size of the JavaScript stack memory in kilobytes. Default value: 32 KB.

Internet Protocol Proxy Node

Enables you to set the proxy server connection settings.

To Access	Vuser > Run-Time Settings > Internet Protocol > Proxy
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
No proxy	All Vusers should use direct connections to the Internet. This means that the connection is made without using a proxy server.

UI Element (A-Z)	Description
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	<p>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.</p> <ul style="list-style-type: none"> ▶ Use automatic configuration script. Allows you to specify a JavaScript file containing proxy assignment information. This script tells the browser when to access a proxy server and when to connect directly to the site, depending on the URL. In addition, it can instruct the browser to use a specific proxy server for certain addresses and another server for other addresses. Specify the location of the script in the Address field. ▶ Use proxy server. You can specify one proxy server for all HTTP sites, and another proxy server for all HTTPS (secure) sites or check the use the same proxy server for all protocols box. ▶ Exceptions Allows you to specify exceptions to the proxy server rules. ▶ Authentication Opens the Proxy Authentication Dialog Box. If the proxy server requires authentication for each Vuser, use this dialog box to enter the relevant password and user name. To add authentication dynamically during recording, or to add authentication for multiple proxy servers, use the web_set_user function. For more information, see the <i>Online Function Reference (Help > Function Reference)</i>.

Java Classpath Node

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper replay.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.

To Access	Vuser > Run-Time Settings > Java Environment Settings > Classpath
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
	Add Classpath. Adds a new line to the classpath list.
	Delete. Permanently removes a classpath.
	Down Arrow. Moves a classpath entry down the list.
	Up Arrow. Moves a classpath entry up the list.
Classpath Entries List	A list of classpath entries.

Java VM Node

Enables you to set the Java VM run-time settings.

To Access	Vuser > Run-Time Settings > Java Environment Settings > Java VM
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Virtual Machine Settings	<ul style="list-style-type: none"> ▶ Use internal logic to locate JDK. Search the PATH, registry, and Windows folder for the JDK to use during replay. ▶ Use specified JDK. Use a specified JDK during replay. <ul style="list-style-type: none"> ▶ JDK. The home directory of the specified JDK. ▶ Additional VM Parameters. Any optional parameters used by the virtual machine. ▶ Using Xbootclasspath parameters. Replays the script with the Xbootclasspath /p option.
Class Loading Settings	<ul style="list-style-type: none"> ▶ Load each Vuser using dedicated class loader. Load each Vuser using a dedicated class loader. This will allow you to use a unique namespace for each Vuser and manage their resources separately.

JMS Advanced Node

Enables you to set the JMS Advanced run-time settings.

To Access	Vuser > Run-Time Settings > JMS > Advanced
------------------	--

Important Information	This node is only available for the Web Service protocol. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.
Relevant Tasks	"Set the Run-time setting - optional" on page 1052

User interface elements are described below:

VM (Virtual Machine)

UI Element (A-Z)	Description
Use external VM	Enables you to select a VM (Virtual Machine) other than the standard one. If you disable this option, users use the JVM provided with VuGen.
JVM Home	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	Vendor implementation of JMS classes together with any other required supporting classes, as determined by the JMS implementation vendor.

JMS

UI Element (A-Z)	Description
Additional VM Parameters	Extra parameters to send to the JVM such as Xbootclasspath, and any parameters specified by the JVM documentation.)
JNDI initial context factory	The fully qualified class name of the factory class that will create an initial context. Select a context factory from the list or provide your own.
JNDI provider URL	URL string of the service provider. For example: <ul style="list-style-type: none"> ➤ Weblogic - t3://myserver:myport ➤ Websphere - iiop://myserver:myport
JMS connection factory	JNDI name of the JMS connection factory. You can only specify one connection factory per script

UI Element (A-Z)	Description
JMS security principal	Identity of the principal (for example the user) for the authentication scheme.
JMS security credentials	The principal's credentials for the authentication scheme.
Number of JMS connections per process	The number of JMS connections per mdrv process, or Vuser . All Vusers sharing a connection will receive the same messages. The default is 1, and the maximum is 50 Vusers . The fewer connections you have per process, the better your performance.
Received message timeout options	<p>The timeout for received messages. The default is No wait.</p> <ul style="list-style-type: none"> ▶ Infinite wait. Wait as long as required for the message before continuing. ▶ No wait. Do not wait for the Receive message, and return control to the script immediately. If there was no message in the queue, the operation fails. ▶ Specify the timeout in seconds. A timeout value for the message. If the timeout expired and no message has arrived, the operation fails. (default) <ul style="list-style-type: none"> ▶ User defined timeout. 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.

Microsoft .NET Environment Node

Enables you to set the .NET run-time settings.

To Access	Vuser > Run-Time Settings > .NET > .NET Environment
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
AUT configuration	<p>Configuration settings for the AUT location and configuration files.</p> <ul style="list-style-type: none"> ➤ 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, enter a full path. If you only specify a file name, and the file is not in the script\bin folder, VuGen loads it from the App base directory.
Concurrency	<p>AppDomain Per Vuser. Enables execution of each Vuser in a separate app domain. Running Vusers in separate App Domains enables each Vuser to execute separately without sharing static variables and prevents locking between them.</p> <p>Default value: true.</p>

MMS Server and Protocol Node

Enables you to set the MMS (Multimedia Messaging Service) run-time settings.

To Access	Vuser > Run-Time Settings > MMS > Server and Protocol
Important Information	<ul style="list-style-type: none"> ▶ This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422. ▶ In the General > Miscellaneous node, under Multithreading, select Run Vuser as a process.

User interface elements are described below:

UI Element (A-Z)	Description
Automatic WAP Connections	<p>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:</p> <p>Per Iteration. Connect at the beginning of each iteration and disconnect at the end of each iteration.</p> <p>Per Send or Receive. Connect and disconnect at the beginning and end of each message.</p> <p>None. Do not use automatic WAP connections.</p> <p>Default value: Per Iteration.</p>
Default Sender address	<p>The default address sent in the Sender header.</p> <p>Default value: +999999.</p>
MMS Version	The version of the MMS protocol used by the script.
MMSC URL	The URL of the MMSC (Multimedia Messaging Center) server.
SMSC IP	The IP address of the SMSC server used for sending MMS notifications over SMPP.

UI Element (A-Z)	Description
SMSC Port	The IP port of the SMSC server used for sending MMS notifications over SMPP.
Timeout (seconds)	The time that the server waits for incoming messages. Default value: 60 seconds.

Network Speed Simulation Node

Enables you to configure the network speed run-time settings.

To Access	Vuser > Run-Time Settings > Network > Speed Simulation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Use bandwidth	Indicate a specific bandwidth level for your Vuser to emulate. You can select a speed ranging from 14.4 to 512 Kbps, emulating analog modems, ISDN, or DSL.
Use custom bandwidth	Indicate a bandwidth limit for your Vuser to emulate. Specify the bandwidth in bits, where 1 Kilobit=1024 bits.
Use maximum bandwidth	Vusers run at the maximum bandwidth that is available over the network. Default value: enabled.

Oracle NCA Client Emulation Node

Enables you to set the Oracle NCA run-time settings.

To Access	Vuser > Run-Time Settings > Oracle NCA > Client Emulation
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Network	<ul style="list-style-type: none"> ➤ Socket Mode. Configure the connection between the client and server at the socket level. ➤ Timeout. The time that an Oracle NCA Vuser waits for a response from the server. Default value: -1 (disables the timeout and the client waits indefinitely). ➤ Pragma Mode. Configure the communication on the Oracle-defined Pragma mode (higher level than socket and HTTP). ➤ Max Retries. The maximum number of IfError messages the client will accept from the server before issuing an error. IfError messages are the periodic messages the server sends to the client, indicating that it will respond with the data as soon as it is able. ➤ Retry Interval (ms). The interval (in milliseconds) between retries in the case of IfError messages. ➤ Include retry intervals in transaction. Includes the interval between retry time, as part of the transaction duration time. ➤ Enable Heartbeat. Sends a heartbeat signal is sent to the server. You can configure the frequency of the heartbeat by setting the frequency property. Default value: enabled, 120.
Connection	<ul style="list-style-type: none"> ➤ Forms version. The version of the Oracle Forms server detected during recording. Modify this setting only if the server was upgraded since the recording. ➤ Command line parameter separator. Marker for the division between separate parameters in a command line string.
Diagnostic	<ul style="list-style-type: none"> ➤ Application Version. The version of Oracle Application. This option is relevant when using Oracle Application—not a custom Oracle NCA application. It is only required when using Oracle database breakdown.

RDP Advanced Node

Enables you to set the RDP advanced run-time settings.

To Access	Vuser > Run-Time Settings > RDP > Advanced
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Bitmap caching	Allows the remote desktop server to use bitmap caching. Enabling this setting can save system resources on the remote desktop server.
Font smoothing	Allows the remote desktop server to use font smoothing. Disabling this setting can save system resources on the remote desktop server.
Menu and window animation	Allows the remote desktop server to animate menus and windows. Disabling this setting can save system resources on the remote desktop server.
Remote desktop composition	Enables remote desktop composition
Show contents of window while dragging	Shows the contents of windows while they are being dragged. Disabling this setting can save system resources on the remote desktop server
Show remote desktop background image	Allows you to run the remote desktop application without displaying the desktop background image on the remote desktop. Disabling this setting can save system resources on the remote desktop server.

UI Element (A-Z)	Description
Socket receive buffer size (bytes)	The number of bytes to allocate for the socket's receive buffer. If the buffer is too small, it can fill up causing the server to disconnect. If the buffer is too large, it uses more local system resources (memory).
Themes	Allows the remote desktop server to use Windows themes. Disabling this setting can save system resources on the remote desktop server.

RDP Agent Node

Enables you to set the RDP Agent run-time settings.

To Access	Vuser > Run-Time Settings > RDP > RDP Agent
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Use RDP agent	Instructs VuGen to use RDP agent during recording, then generates script using information gathered by the RDP agent during the recorded session. LoadRunner RDP agent must be installed on the server.
Enable RDP agent log	<p>Enables the RDP agent log. This feature is should be used only for debugging purposes.</p> <ul style="list-style-type: none"> ➤ RDP agent log detail level. Configures the level of detail generated in the RDP agent log with Standard being the lowest level of detail and Extended Debug being the highest level of detail. ➤ RDP agent log destination. Configures the destination of the RDP agent log data. File saves the log messages only on the remote server side. Stream sends the log messages to the Vugen machine. FileAndStream sends the log messages to both destinations. ➤ RDP agent log folder. The folder path on the remote server that the RDP agent log file will be generated in. If none is specified and the agent log destination was set to File, the log is saved in the temp folder of the user on the server.

RDP Configuration Node

Enables you to set the RDP configuration run-time settings.

To Access	Vuser > Run-Time Settings > RDP > Configuration
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Enable RDP caching	Support data caching orders in RDP (enabled by default)
RDP Client Version Emulation	The version of RDP packets to produce during replay: As Recorded , or a specific version number.
Remote desktop color depth	The color depth settings for the replay: As Recorded , or a specific depth
Remote desktop resolution (pixels)	The size of the window in which the applications are run: As Recorded , or a specific size (in pixels)
Start the following program on connection	Open RDP connection to invoke the specified application. Specify the following information: Program path and file name and optionally, Start in folder

RDP Synchronization Node

Enables you to set the RDP synchronization run-time settings.

To Access	Vuser > Run-Time Settings > RDP > Synchronization
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Default input origin	The default origin for input operations.
Default offset addition.	Saves the offset of images that moved during synchronization for all subsequent functions. Default value: No.
Default synchronization timeout (sec)	The time in seconds to wait for synchronization operations. Enter a value between 0 and 1000. Default value: 60.
Default tolerance for image synchronization	The tolerance level for performing synchronization on images. Select one of the options: Exact , Low , Medium , or High . High has the most tolerance for changes and mismatches. Low requires a match of approximately 95 percent, Medium requires a match of approximately 85 percent, High requires a match of approximately 70 percent, and Exact requires an 100 percent match. Default value: Medium.
Disable synchronization failure dialog	When selected, it prevents the Synchronization Failure Dialog box from opening. Default value: not selected.

UI Element (A-Z)	Description
Fail image synchronization step on timeout	Instructs Vusers how to proceed when images are not found during synchronization. Yes sets a Fail status and Vusers follow the Continue on Error setting. No returns an LR_NOT_FOUND flag, the step reports a warning and the script continues. Default value: Yes.
Recorded	Uses coordinates for all input operations with a non-specified input origin. Default value: enabled.
Synched	Adds the most recent offsets saved at one of the previous synchronization functions to the recorded coordinates of each input operation with a non-specified input origin.
Typing speed (msec/char)	The time in milliseconds for sending consecutive characters in keyboard commands. Enter a value between 0 and 1000. Default value: 150.

RTE Node

Enables you to set the RTE run-time settings.

To Access	Vuser > Run-Time Settings > RTE > RTE
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Maximum number of connection attempts	<p>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.</p> <p>Default value: 5.</p>
Use original device name	<p>In certain environments, each session (Vuser) requires a unique device name. The TE_connect function generates a unique 8-character device name for each Vuser, and connects using this name. Select this option to connect using the device name that is contained within the com_string parameter of the TE_connect function.</p> <p>Note: The original device name setting applies to IBM block-mode terminals only.</p>
Delay before typing	<p>The delay setting determines how Vusers execute TE_type functions.</p> <ul style="list-style-type: none"> ▶ First key. Specify the amount of time (in milliseconds) that a Vuser waits before entering the first character in a string. ▶ Subsequent keys. Specifies the amount of time (milliseconds) that a Vuser waits before between submitting successive characters. <p>Note: You can use the TE_typing_style function to override the Delay settings for a portion of a Vuser script.</p>
X-System synchronization	<ul style="list-style-type: none"> ▶ Timeout. The timeout (in seconds) to wait for the system to stabilize when replaying a TE_wait_sync function before an error is returned. ▶ Stable time. The time (in milliseconds) that the Vuser waits to ensure that the terminal is no longer in the X-SYSTEM mode after executing a TE_wait_sync function.

SAPGUI General Node

Enables you to set the SAPGUI run-time settings.

To Access	Vuser > Run-Time Settings > SAPGUI > General
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Send status bar text	Send the text from the status bar to the log file.
Send active window title	Send the active window title text to the log file.

UI Element (A-Z)	Description
<p>Show SAP Client during replay</p>	<p>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.</p> <ul style="list-style-type: none"> ▶ 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 "How to Enhance SAPGUI Scripts" on page 851.
<p>Options...</p>	<p>Opens the SAPGUI Advanced Options Dialog Box, enabling you to set the following settings:</p> <ul style="list-style-type: none"> ▶ 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. <ul style="list-style-type: none"> ▶ Timeout to SAPfewgsvr. The SAPfewgsvr.exe process timeout in seconds. Default value: 300 seconds.

Silverlight Services Node

Displays the WSDL files associated with your script and allows you to modify their settings for the replay phase.

To Access	Vuser > Run-Time Settings > Silverlight > Services
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
<Service List>	A list of the WSDL files that are available for this script.
Protocol & Security Data	Opens the Protocol and Security Scenario Data dialog box, allowing you to configure a number of settings for each selected WSDL. For more information, see "Protocol and Security Scenario Data Dialog Box" on page 412.

VBA Node

Enables you to set the Visual Basic run-time settings.

To Access	Vuser > Run-Time Settings > VBA > VBA
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
VBA References List	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.
VBA Compiler Options	<ul style="list-style-type: none"> ▶ Debug script through VBA IDE. Enables debugging through the Visual Basic IDE (Integrated Development Environment). ▶ On Error keep VBA IDE visible. Keeps the Visual Basic IDE visible during script execution.

WAP Gateway Node

Enables you to set the WAP gateway run-time settings.

To Access	Vuser > Run-Time Settings > WAP > Gateway
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
HTTP Direct	Run the Vusers run in HTTP mode, accessing a Web server directly.
WAP Gateway Property	<p>Run the Vusers accessing a Web server via a WAP Gateway.</p> <ul style="list-style-type: none"> ▶ IP. The IP of the gateway. ▶ Port. The port of the gateway. When running your Vusers through a WAP gateway, VuGen automatically sets default port numbers, depending on the selected mode. However, you can customize the settings and specify a custom IP address and port for the gateway. ▶ WAP 1.x (WSP). Selects the appropriate WAP version. If you recorded in WAP 1.x (WSP), you can run the Vuser in either 1.x (WSP), or 2.0 (HTTP proxy) mode. If you select this option, you can set the WAP 1.x (WSP) Properties. For more information, see below. ▶ WAP 2.0 (HTTP). Selects the appropriate WAP version. If you recorded in WAP 2.0 (HTTP proxy), then you can only run the Vuser in the same mode.

WAP 1.x (WSP) Properties

UI Element (A-Z)	Description
Advanced	Expand to set the Advanced Properties. For more information see below.
Connection Options	<ul style="list-style-type: none"> ▶ Connection-oriented Mode sets the connection mode for the WSP session to Connection-Oriented. ▶ Connectionless Mode sets the connection mode for the WSP session to Connectionless.
Enable Security	Enable a secure connection to the WAP gateway.

Advanced Properties

UI Element (A-Z)	Description
Acknowledge headers	Returns standard headers that provide information to the gateway. Default value: disabled.
BearerType	The type of bearer used as the underlying transport.
CAPSessionResume	Enables requests for session suspend or resume.
Client SDU buffer size	The largest transaction service data unit that may be sent to the client during the session. Default value: 4000.
Confirm Push support	In CO mode, if a push message is received, this option instructs the Vuser to confirm the receipt of the message. For more information, see "VuGen Push Support" on page 1195.
MethodMOR	The number of outstanding methods that can occur simultaneously.
Network MTU Size	The maximum size in bytes, of the network packet. Default value: 4096.
Push support	Enables push type messages across the gateway. Default value: disabled.
PushMOR	The number of outstanding push transactions that can occur simultaneously.
Retrieve messages	When a push messages is received, this option instructs the Vuser to retrieve the message data from the URL indicated in the push message. Default value: disabled.
Server SDU buffer size	The largest transaction service data unit that may be sent to the server during the session. Default value: 4000.
Support Cookies	Provide support for saving and retrieving cookies. Default value: disabled.

UI Element (A-Z)	Description
WTLS Abbreviated Handshake	Use an abbreviated handshake instead of a full one, when receiving a redirect message. Default value: False.
WTLS Deffie Hellman	Use the Deffie Hellman encryption scheme for WTLS (Wireless Transport Layer Security) instead of the default scheme, RSA. Default value: False.
WTLS Deffie Hellman identifier	An identifier for the Deffie Hellman encryption scheme. This identifier is required for the abbreviated handshake with the Operwave gateway that uses the Deffie Hellman encryption scheme.
WTP Retransmission Time	The time in seconds that the WTP layer waits before re-sending the PDU if it did not receive a response. Default value: 5000.
WTP Segmentation and Reassembly	Enables segmentation and reassembly (SAR) in WTP, Wireless Transport Protocol. Default value: True.

WAP Radius Node

Enables you to set the WAP radius run-time settings.

To Access	Vuser > Run-Time Settings > WAP > Radius
Important Information	This node is only available for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 422.

User interface elements are described below:

UI Element (A-Z)	Description
Accounting port number	Accounting port of the Radius server.
Authentication port number	Authentication port of the Radius server.
Connection Timeout (sec)	The time in seconds to wait for the Radius server to respond. Default value: 120 seconds.
IP Address	IP address of the Radius server.
Network Type	Accounting network type: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).
Radius client IP	Radius packets source IP, usually used to differentiate between packets transmitted on different NIC cards on a single Load Generator machine.
Retransmission retries	The number of times to retry after a failed transmission. Default value: 0.
Secret Key	The secret key of the Radius server.
Store attributes returned by the server to parameters	Allow Vusers to save attributes returned by the server as parameters, which can be used at a later time. Default value: False.

Part II

Protocols



12

AJAX Protocol

This chapter includes:

Concepts

- ▶ [AJAX Protocol Overview on page 500](#)
- ▶ [AJAX Supported Frameworks on page 500](#)
- ▶ [AJAX Example Script on page 501](#)

Concepts

AJAX Protocol Overview

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.

AJAX Supported Frameworks

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

AJAX Example Script

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported AJAX controls, VuGen generates a function with an **ajax_xxx** prefix.

In the following example, a user selected item number 1 (index=1) in an Accordion control. VuGen generated an **ajax_accordion** function.

```
web_browser("Accordion.aspx",
            DESCRIPTION,
            ACTION,
            "Navigate=http://labm1app08/AJAX/Accordion/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.

13

Ajax TruClient Protocol

This chapter includes:

Concepts

- ▶ Ajax TruClient Protocol Overview on page 504
- ▶ Script Levels on page 506
- ▶ TruClient Snapshots on page 507
- ▶ Alternative Steps on page 508
- ▶ Global Firefox Browser Settings on page 508
- ▶ Tips and Tricks on page 509
- ▶ Firefox Private Browsing on page 509

Tasks

- ▶ How to Record Ajax TruClient Scripts on page 511
- ▶ How to Enhance Ajax TruClient Scripts on page 514
- ▶ How to Debug Ajax TruClient Scripts on page 517
- ▶ How to Resolve Object Identification Issues on page 521
- ▶ How to Insert and Modify Loops on page 523
- ▶ How to Insert Custom JavaScript and C Code into Ajax TruClient Scripts on page 524

Reference

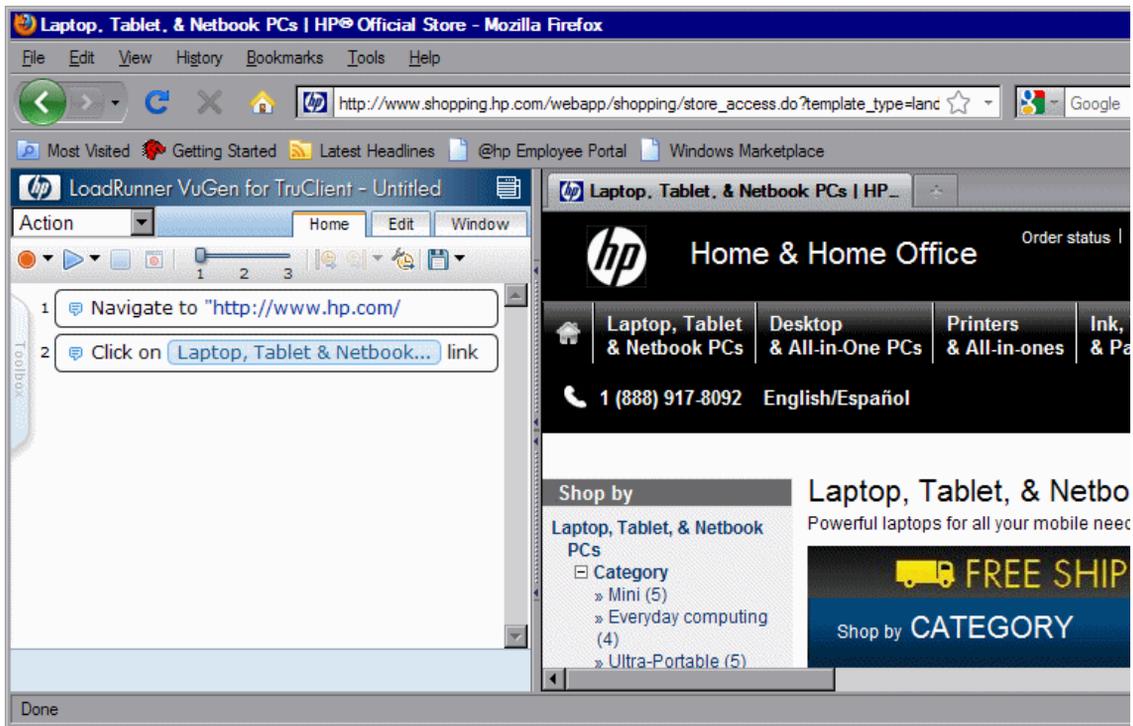
- ▶ TruClient Step Arguments on page 526
- ▶ LoadRunner Functions on page 530
- ▶ Ajax TruClient User Interface on page 532

TruClient Limitations on page 540

Concepts

Ajax TruClient Protocol Overview

The Ajax TruClient protocol interactively records scripts on the user level. This enables VuGen to record dynamic, complex web-based applications and create user friendly scripts. Scripts are created in real-time and steps can be seen in the LoadRunner VuGen for TruClient tab of Mozilla Firefox as they are performed.



For user interface details, see "Ajax TruClient User Interface" on page 532.

You can watch a video demonstration from the VuGen start page. The workflow is completely new and most of it is different from the workflow for other VuGen protocols. The following lists some of the main differences between the Ajax TruClient protocol and other VuGen protocols:

- ▶ The script is not visible in VuGen's script view. It is created and modified in the VuGen tab in Mozilla Firefox.
- ▶ TruClient scripts are asynchronous. Steps do not have to wait for previous steps to complete. Each step defines an **End Event** which defines the point at which subsequent steps are allowed to start running.
- ▶ TruClient scripts are recorded on the user level, therefore there are no correlations.
- ▶ All events during recording are saved in the script. Events deemed to be irrelevant are assigned to different script levels and are not replayed unless the level is manually changed by the user.
- ▶ TruClient transactions are defined by step events, not the steps themselves as in other protocols. For example, a step's End Event may allow the script to continue, while a transaction that ends on that step may continue until the step event that defines the transaction is reached.
- ▶ The Run-Logic in TruClient scripts is controlled differently. There is only one action.
- ▶ TruClient step arguments accept JavaScript code as values.

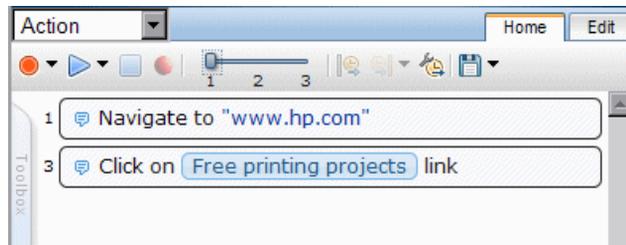
Most of the tasks involved in recording, replaying, and modifying scripts is done using the LoadRunner VuGen for Truclient tab of Mozilla Firefox.

Script Levels

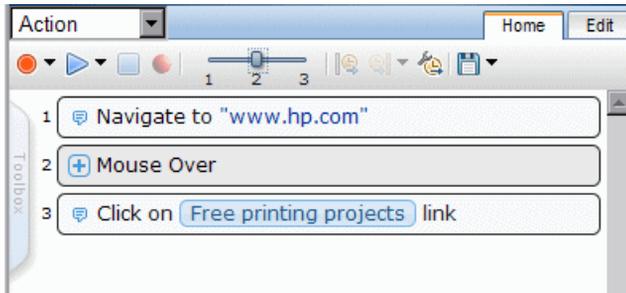
As part of the process of recording a business process, some steps that are performed by the user while recording are not needed during replay. VuGen removes steps it deems to be unnecessary and places them in different script levels. For example, a click step that occurs in an area of the application that has no effect is placed in level 2. VuGen assumes that this step is not significant and will not help the user to emulate a business process on the application. During the replay phase, only steps that are visible are run. The default view displays level 1 steps only. To view steps from levels 2 and 3 as well, use the slide bar in the home tab.

In certain cases, you may want to override VuGen's decisions and manually change the level of a given step. This can happen in cases such as mouse over steps that are needed during replay. VuGen generally views mouse over steps as unnecessary for replay and assigns them to level 3. For more information, see "Modify and view script levels" on page 519.

The following screen shot displays a small script. Note that the step numbers skip from 1 to 3. Step 2 is hidden in a different level.



After changing the display settings by using the slide bar, all steps are now displayed and will run if replayed in interactive mode.



TruClient Snapshots

TruClient automatically generates snapshots during recording. These snapshots can be viewed by hovering the mouse over each step's icon. The snapshots are taken before the step's action is implemented. They are saved as .png files. Click each snapshot to display it in a new Firefox tab. Make sure that the correct tab is active before replay. Recording snapshots are stored in the snapshot directory.

TruClient can also generate snapshots during replay and load mode according to your specifications in the Run-Time settings. For more information, see "General Other Settings Node" on page 444.

Replay snapshots are stored in the results directory and are organized according to the type of replay (interactive or load), the script section, and the iteration.

Note: The default snapshot locations may change in future releases.

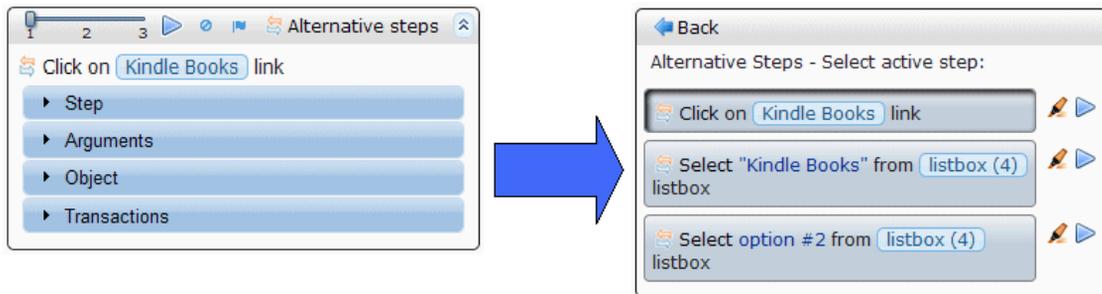
Alternative Steps

Alternative steps allow you to view instances in which there are multiple ways to perform the same action in a step. You can modify such steps to perform the given action to debug or enhance your script. For example, in a drop-down list, VuGen gives you the option of specifying your selection by name or by the number in which it appears in the list.



Steps that have alternative options are labeled with an alternative step symbol. Click it to view the alternative options for that step. Click the desired alternative and select **Back**.

Below is a snapshot of a step in which the second item in a drop-down list named "Kindle Books" was selected. The alternative steps feature gives you the option of defining the step based on clicking the link "Kindle Books", selecting the object "Kindle Books" from the drop-down menu, or selecting the second item in the drop-down menu.



Global Firefox Browser Settings

Each TruClient script is opened in Firefox with a different profile. Firefox profiles save user data such as cookies, client certificates, history, cache, etc. To make changes that are saved in the script profile, make the changes in the Firefox window when interactively developing the script. These changes will apply to the current script only.



To make changes that apply to all scripts, open the Browser Configuration settings dialog box, click the Edit Browser Options button from the Record toolbar in the VuGen main window. Most of the settings in this dialog box are imported to each new script as it is created. The Firefox extensions settings are imported to each script every time a script is opened in Firefox. The proxy settings can be imported via the Run-Time Settings dialog box **General > Other Settings** node.

Tips and Tricks

The following section contains tips and tricks for recording Ajax TruClient scripts.

- ▶ Do not change the size of the application window between recording and replaying your script. It can cause objects to move and interfere with VuGen's ability to locate them.
- ▶ Do not use the arrow keys, the tab key, the escape key, or the middle mouse button when recording.
- ▶ In certain cases where applications are sensitive to the focus status certain steps may have problems replaying. This may occur due to an object that either captures the focus or requires a certain focus. In either case, you can try clicking on another object to change the focus state immediately before the problematic step.
- ▶ For more tips and tricks, see http://h30501.www3.hp.com/t5/Best-Practices-and-Methodology/bd-p/APP_Perf_Val_BP.

Firefox Private Browsing

Private Browsing is a Mozilla Firefox mode which allows a user to browse without saving information about their session. Some examples of items which are not saved are passwords, cookies, and history.

To more accurately emulate real users, VuGen replays scripts in private browsing mode. This ensures that the browser does not use saved session information when running a script more than one time.

You can manually change enable and disable private browsing from the Firefox menu.

Tasks

How to Record Ajax TruClient Scripts

This task describes the basic steps involved in interactively recording an Ajax TruClient Vuser Script.

This task includes the following steps:

- "Configure the Run-Time Settings" on page 512
- "Configure the Global Browser Configuration Settings" on page 512
- "Start developing the script" on page 512
- "Record interactively" on page 512
- "Enhance the script" on page 513
- "Replay the script in Firefox" on page 513
- "Stop developing" on page 513
- "Replay the script in Load Mode" on page 513

1 Configure the Run-Time Settings

Configure the Run-Time settings before recording and performing a load test. To open the Run-Time settings dialog box, click F4. For more information, see "Run-Time Settings" on page 417.

2 Configure the Global Browser Configuration Settings

The Firefox Browser Configuration settings allow you to configure settings that apply to all TruClient scripts. The settings are imported to new scripts as they are created. For more information, see "Global Firefox Browser Settings" on page 508. To open the Browser Configuration settings dialog box, click the Edit Browser Options button from the Record toolbar in the VuGen main window.



3 Start developing the script

Click **Develop Script** to initialize the interactive recording session in Mozilla Firefox.

4 Record interactively

Navigate to the desired starting website and click record. All of your actions will be recorded and displayed in the VuGen tab on the left as you perform your business process. You can pause or stop the script and continue recording from any point in the script.

To record into different sections of the script, use the drop down bar above the toolbars.

5 Enhance the script

You can enhance your script in a number of ways such as inserting parameters, transaction, loops, and verification steps. For task details, see "How to Enhance Ajax TruClient Scripts" on page 514.

6 Replay the script in Firefox

Replay the script at least two times, correcting any errors that occur during the process. After two successful consecutive replays, you can move on to the next step. If you continue to experience errors, see "How to Debug Ajax TruClient Scripts" on page 517.

7 Stop developing



Click the Save button to save the script. Close the firefox window.

8 Replay the script in Load Mode

TruClient scripts are run slightly differently when performing load testing, so Load Mode was created to run the script exactly as it will run during load testing. In the VuGen main window, click the arrow next to the Develop Script button to replay the script in Load Mode. Progress can be monitored in the Interactive Replay log. Firefox does not open, and snapshot are not displayed.

Note: Any customizations (such as bookmarks) that you make within this instance of Firefox will not be saved globally. This is because VuGen opens each script in a unique Firefox profile. If you want to use firefox for any use other than creating this script (e.g. browse the internet), we recommend that you open an additional firefox window.

How to Enhance Ajax TruClient Scripts

There are a number of optional enhancements that can be added to scripts beyond the basic workflow. This task describes the enhancements and how to use them.

This task includes the following steps:

- "Modify Steps" on page 515
- "Insert loops" on page 515
- "Insert If blocks and exit steps" on page 515
- "Insert comments" on page 516
- "Insert Transactions" on page 516
- "Create Parameters" on page 516
- "Insert Catch Error Steps" on page 517
- "Verify that an objects exist" on page 517
- "Insert Generic Steps" on page 517

Modify Steps

Modify step arguments and objects by selecting the desired step and expanding the options. This expands the step and allows you to modify the objects and properties. For a detailed list of the step structure, see "TruClient Step Structure" on page 537.

Insert loops

Loops repeat selected portions of the script until certain criteria is met or for a specified number of times. To insert a loop, select **Toolbox > Flow Control > For loop**. For more information, see "How to Insert and Modify Loops" on page 523.

Insert If blocks and exit steps

To conditionalize a portion of the script, you can insert If blocks. To insert an If block, select **Toolbox > Flow Control > If block**.

Exit steps cause a script to exit the iteration or the entire script. These can be used with If statements to exit a script or iteration when a specified condition occurs. To insert an exit step, select **Toolbox > Flow Control > Exit**.

Insert comments

You can insert comments into your script by selecting **Toolbox > Misc** and dragging the **Comment** icon to the desired location.

Insert Transactions



You can add transactions by using the Transaction Editor. To open the Transaction Editor click the Transaction Editor button from the Home Tab or click Ctrl + Alt + F7. TruClient transactions function differently from other protocols because of the asynchronous nature of TruClient steps. Transactions are defined based on start and end steps and step events. Due to this definition, a transaction's end can be triggered before the true end of a step.

Create Parameters

Parameters for Ajax TruClient scripts can be created in the standard method for all protocols. For detailed information about VuGen parameters, see "Parameters" on page 257.

Parameters can be referenced and created within step arguments or Eval JavaScript steps by using the following syntax.

```
LR.getParam(paramname)
```

Returns the next value of the parameter **paramname**.

```
LR.setParam(paramname, value1)
```

Updated the value of an existing parameter with the value **value1**.

Insert Catch Error Steps

Catch error steps are group steps that run their contents if the previous step contains an error. Additionally, the error is "caught" and is not returned. You can define catch error steps to catch any error, or a specific type of error. If there are two catch error steps in a row, they both apply to the same step. To insert a catch error step, select **Toolbox > Flow Control > Catch Error**.

Verify that an objects exist

To verify that a string or object exists in the application, you can insert a verify step:

- a** Select **Toolbox > Functions** and drag the **Verify** icon to the desired location.
- b** Click the object in the verify step.
- c** Select the object you want to verify.

Insert Generic Steps

You can insert a blank step and manually configure it. To insert a generic step, select **Toolbox > Functions > Generic Object/Browser Action**, expand the step, and enter the desired step properties. Generic Object Actions perform an unspecified action on an object. Generic Browser Actions perform an unspecified action on the browser such as go back, reload, switch tabs, etc.

How to Debug Ajax TruClient Scripts

This task describes the basic steps involved in interactively recording an Ajax TruClient Vuser Script.

This task includes the following steps:

- "View Replay Errors in Firefox" on page 518
- "Run The Script Step by Step" on page 518
- "View the Replay Logs" on page 518
- "Insert Breakpoints" on page 518

- "Debug Scripts Using Snapshots" on page 518
- "Modify and view script levels" on page 519
- "Insert Wait steps" on page 519
- "View the Tips and Tricks Guide" on page 520

View Replay Errors in Firefox



If any steps failed during replay, they are marked with an error icon. Hover the mouse over these icons to view descriptions of the errors.

Run The Script Step by Step

You can run your script step by step to view the replay more slowly and in a controlled manner. To run the script step by step, select the down arrow from the replay button in Firefox and select **Replay step by step**. Repeat this procedure after each step to continue the step by step replay.

View the Replay Logs

In the VuGen's Output Window, you can view details your script's replay in the Replay and Interactive Replay Logs. For more information, see "Output Window" on page 79.

Insert Breakpoints

Breakpoints instruct the script to stop running during a replay when in interactive mode. They can be used to help debug your script. To insert a breakpoint, select the desired step and click the Breakpoints button.



Debug Scripts Using Snapshots

You can use the snapshots generated during replay to debug scripts by viewing the snapshots of the failed step(s).

- a** Open the Run-Time settings dialog box and go to the **General > Other Settings** node.
- b** Set the **Replay Snapshot Generation** setting to **On Error**.
- c** Replay the script.

- d Look in the Replay or Interactive Replay logs for errors. Note the step numbers of the steps that had errors.
- e In the VuGen main window, right-click and select **Open Script Directory > Results > Interactive** and select the relevant section and iteration.

You now have a group of snapshots in which errors occurred in the script.

Modify and view script levels

Sometimes, steps that were recorded and are necessary for replay are placed in levels 2 and 3. In this case, you need to manually modify the level of those steps to level 1.

- ▶ To modify a the script's replay level, drag the slider in the toolbar to the desired level. Dragging the slider to level 3 displays and replays the steps on levels 1, 2, and 3.
- ▶ To move a step to a different level, open the step and click on the step section. Move the slider to the desired level. If the step is part of a group step, both the group step and the individual step must be modified.

For more information, see "Script Levels" on page 506.

Insert Wait steps

Wait steps cause the script to pause for a specified amount of time before continuing with the next step. Wait for Object steps cause the script to wait for a specified object to load before continuing with the next step. Wait steps differ from think time steps in other protocols because the dynamic nature of the TruClient protocol. Wait steps begin after the **End Event** of the previous step is reached. This means that the previous step may continue to

run after the wait step has been reached. To insert a wait, select **Toolbox > Functions** and drag the **Wait** or **Wait for Object** icon to the desired location in your script. Wait steps wait for a specified amount of time, Wait for Object steps wait until the specified object appears in the application. In Wait for Object steps, select the **Click to choose an object** button to select the target object in the application.

View the Tips and Tricks Guide

For more helps and hints, see http://h30501.www3.hp.com/t5/Best-Practices-and-Methodology/bd-p/APP_Perf_Val_BP

How to Resolve Object Identification Issues

In dynamic websites, objects which have been recorded can often move or change content. This can cause the script to lose the ability to locate the object. The following steps describe the ways to resolve these issues. When identifying objects for applications that recorded in windows, make sure that the correct window is selected using the Window tab.

- "Highlighting an object" on page 521
- "Improve Object Identification" on page 521
- "Modify the Object Identification Method" on page 521
- "Modify the script timing" on page 522
- "Relating objects to other objects" on page 523
- "Replacing an object" on page 523

Highlighting an object



Regardless of which method of object identification is used, you can use the highlight button to check if an object is visible in the application at any time. If the object cannot be found, an error message is displayed.

Improve Object Identification



Use this option first if the object was not found or if multiple objects were found. Objects are identified using the objects properties. Some of these properties may be dynamic and thus prevent object identification during replay. The Improve Object Identification button next to the ID Method field identifies the object a second time and uses the two definitions to create a more accurate object definition.

Modify the Object Identification Method

You can modify the way TruClient identifies the object by modifying the object identification method in the Object section of the step properties. defined The following options are available:

- **Automatic.** TruClient's default object identification method. If this method does not successfully find the object during replay, click the

Improve Object Identification button and replay the script again.

- ▶ **XPath.** Identifies the object based on an xpath expression that defines the object in the DOM tree. You can manually modify the expression. To regenerate the original expression generated by VuGen, click the Regenerate Expression button.
- ▶ **JavaScript.** JavaScript code that returns an object. For example: `document.getElementById("SearchButton")` returns an element that has a DOM ID attribute of "SearchButton".

Modify the script timing

Sometimes objects may not be found because of timing and synchronization issues. For example, the script may be looking for an object that was in the application, but the script replayed too quickly and already progressed to another page. If you suspect that the object is not being found because of a timing or synchronization issue, you can insert Wait steps. For more information, see "Insert Wait steps" on page 519.

Relating objects to other objects

If an object becomes difficult to identify on its own, you can label the object based on a different, more stable object. For example, you can select an object which is not dynamic and "relate it" to the target object. Relations are defined visually, relating objects according to their distance in pixels from other objects. Relations are defined per ID method, per object. If more than one relation is defined for an ID method of a given object, both relations must locate the same object for the step to pass. VuGen then uses this object to help locate the target object. To use this function, expand the step, select **Object > Related Objects**, and click the add button. Follow the directions to create a relation. Verify that it has worked by highlighting both the object and its related object.



Replacing an object

If you selected the wrong object during recording, or an object has permanently changed you can replace it with a different object without replacing the step. This effectively resets the step, deleting changes made to the original step such as relations. Expand the step, select **Object**, and click the Replace button. Select the new object and replay the script.



How to Insert and Modify Loops

Loops repeat selected portions of the script until certain criteria is met or for a specified number of iterations. You can insert loops and loop modifiers from the **Functions** section of the **Toolbox**.

- "For Loops" on page 524
- "Break statements" on page 524
- "Continue statements" on page 524

For Loops

For loops perform the steps surrounded by the loop until the end condition is met or the code reaches a break statement. Loops arguments use JavaScript syntax. To insert a for loop, select **Toolbox > Functions > For Loop**.

Break statements

Break statements indicate that the current loop should end immediately. For example, if a break statement is encountered in the second of five iteration in a for loop, the loop will end immediately without completing the remaining iterations. To insert a break statement, select **Toolbox > Functions > Break**.

Continue statements

Continue statements indicate that the current loop iteration should end immediately. The loop condition is then checked to see if the entire loop should end as well. For example, if a continue statement is encountered in the second of five iterations in a for loop, the second iteration will end immediately and the third iteration will begin. To insert a continue statement, select **Toolbox > Functions > Continue**.

How to Insert Custom JavaScript and C Code into Ajax TruClient Scripts

This task describes how to insert code into an Ajax TruClient script. You can insert code into a pre-existing step as part of a step argument or insert steps that are completely comprised of external code (C or JavaScript).

This task includes the following steps:

- "Insert code into a pre-existing step" on page 525
- "Insert steps composed entirely of code" on page 525

1 Insert code into a pre-existing step

You can insert JavaScript code into pre-existing steps in the arguments fields of most steps. This allows you to perform any number of customizations.

2 Insert steps composed entirely of code

You can insert steps comprised entirely of code into your script. To do so, select **Toolbox > Function** and drag the **Eval Javascript**, **Eval C**, or **Eval JS on Object** icon to the desired location. The **Eval JS on Object** step runs to the JavaScript code after the specified Object has loaded. We recommend avoiding Eval C and using JavaScript instead wherever possible. The user can refer to this object as the variable "object" in the JavaScript code within the step.

Example:

The following code creates a variable called amount that generates a random number between 1 and 5. You can then use this variable in the argument fields of other steps.

```
var amount=Math.floor(Math.random()*5)+1;
```

Reference

TruClient Step Arguments

The following table displays the step arguments categorized by role. Mandatory arguments are marked with a red star to the left of the argument name in the user interface. All arguments can accept JavaScript code and LoadRunner functions as values. For a list of LoadRunner functions, see "LoadRunner Functions" on page 530.

Role	Action	Arguments
element	Evaluate JavaScript	Code: JavaScript code
element	Mouse Actions: Mouse Down, Mouse Up, Mouse Over, Click, Double Click	<ul style="list-style-type: none"> ▶ Button: The mouse button that is clicked. ▶ X Coordinate: The offset location of the action relative to the upper left corner of the object. This number must be positive. If not specified, the default is the center of the object. ▶ Y Coordinate: The offset location of the action relative to the upper left corner of the object. If not specified, the default is the center of the object. ▶ Ctrl Key: Whether or not this key is pressed during the action. ▶ Alt Key: Whether or not this key is pressed during the action. ▶ Shift Key: Whether or not this key is pressed during the action.

Role	Action	Arguments
element	Drag	<ul style="list-style-type: none"> ➤ Button: The mouse button that is clicked. ➤ X Offset: The amount of pixels to drag the object on the x axis. A positive number indicates a drag to the right. ➤ Y Offset: The amount of pixels to drag the object on the y axis. A positive number indicates a drag down. ➤ Path: List of coordinates representing user drag path. Do not modify this argument.
element	Drag To	<ul style="list-style-type: none"> ➤ Target Object: The step object is dragged to this target object. ➤ X Offset: The offset from the top left of the target object in the x axis. This number must be positive. ➤ Y Offset: The offset from the top left of the target object in the y axis. This number must be positive.
element	Verify	<ul style="list-style-type: none"> ➤ Value: The string or number to verify. ➤ Property: The object property in which to verify the value: <ul style="list-style-type: none"> ➤ Visible text - items that are visible in the application. ➤ All text - items that are in the application but are not necessarily visible. Items in this category are contained in DOM property textContent. ➤ Inner HTML - items contained in the DOM property innerHTML. ➤ Condition: The relationship between the value and property arguments.
focusable	Press Key	<ul style="list-style-type: none"> ➤ Key name: Enter or Space.

Role	Action	Arguments
text box	Type	<ul style="list-style-type: none"> ▶ Value: What is typed. ▶ Clear: Clear the textbox before typing. The default is true. ▶ Typing Interval: The average time in milliseconds between keystrokes.
checkbox	Set	<ul style="list-style-type: none"> ▶ Checked: Set the checkbox to either checked (T) or unchecked (F).
listbox	Select	<ul style="list-style-type: none"> ▶ Text: The selected string. ▶ Ordinal: The order of the selected item in the list. If the text argument is also specified, than this argument refers to the instance of the specified text value in the listbox. An ordinal of 0 generates a random value.
radiogroup	Select	<ul style="list-style-type: none"> ▶ Text: The selected string. ▶ Ordinal: The order of the selected item in the list. If the text argument is also specified, than this argument refers to the instance of the specified text value in the listbox. An ordinal of 0 generates a random value.
filebox	Set	<ul style="list-style-type: none"> ▶ Path: The selected path.
slider	Set	<ul style="list-style-type: none"> ▶ Value: The value that the slider is set to.
datepicker	Set Day	<ul style="list-style-type: none"> ▶ Day: An integer between 1-31 representing the day of the month.
browser	Activate	<ul style="list-style-type: none"> ▶ Ordinal: Defined as an integer. Moves the specified browser window to the foreground.
browser	Activate Tab	<ul style="list-style-type: none"> ▶ Ordinal: Which tab (integer) to activate.
browser	Close Tab	<ul style="list-style-type: none"> ▶ Ordinal: Which tab (integer) to close.
browser	Add Tab	<ul style="list-style-type: none"> ▶ Location: The URL to navigate to in the newly opened tab.
browser	Navigate	<ul style="list-style-type: none"> ▶ Location: The URL to navigate to.
browser	Go Back	<ul style="list-style-type: none"> ▶ Count: The number of pages to go back.

Role	Action	Arguments
browser	Go Forward	<ul style="list-style-type: none"> ➤ Count: The number of pages to go forward.
browser	Resize	<ul style="list-style-type: none"> ➤ Width: The new width. Leaving this blank means do not resize the width. ➤ Height: The new height. Leaving this blank means do not resize the height.
browser	Scroll	<ul style="list-style-type: none"> ➤ X Coordinate: The new x coordinate. Leaving this blank means do not scroll along the x axis. ➤ Y Coordinate: The new y coordinate. Leaving this blank means do not scroll along the y axis.
browser	Dialog - Confirm	<ul style="list-style-type: none"> ➤ Button: Ok or Cancel.
browser	Dialog Prompt	<ul style="list-style-type: none"> ➤ Value: The string to enter. ➤ Button: Ok or Cancel.
browser	Dialog - Authentication	<ul style="list-style-type: none"> ➤ Username: The username to enter. ➤ Password: The password to enter. ➤ Domain: The domain to enter. ➤ Button: Ok or Cancel.

LoadRunner Functions

The following functions can be inserted as values in TruClient step elements.

Method	Description	Arguments	Related Function
The arguments in this row can be used in all methods		<ul style="list-style-type: none"> ▶ Object. The step's object as defined in the application. ▶ Window. Points to the global window object of the application. ▶ Document. The global document object of the application. 	
LR.setParam(name, value)	Saves a string to a parameter, creating the parameter if it does not exist.	<p>name. The name of the parameter in which to save the value.</p> <p>value. The value.</p>	lr_save_string
LR.getParam(name)	Returns the value of the specified parameter.	name. The parameter name.	lr_eval_string
LR.getLRAttr(name)	Returns the value of the specified mdrv command parameter.	name. The name of the command-line parameter.	lr_get_attr_*

Method	Description	Arguments	Related Function
LR.evalC(fun cname, filename)	Runs the specified function whose definition is found in the specified file.	funcname. The function name. filename. The file in which the function is defined. Relative to the script location. If not specified, the default is use (C-functions.c).	None
LR.log(text, level)	Logs a message	text. The message. level. One of the following: <ul style="list-style-type: none"> ➤ "Error", ➤ "Warning", ➤ "Standard", ➤ "Extended", ➤ "Status". example: LR.log("text", "Error");	lr_debug_message
LR.decrypt(te xt)	Returns the text after decryption.	text. The encrypted text.	lr_decrypt
LR.userDataP oint(name, value)	Records a user-defined data point for analysis.	name. The name of the data point. Do not begin a data-point name with any of these strings: HTTP, NON_HTTP, RETRY, mic_, stream_, mms_ value. The numeric value.	lr_user_data_point

Method	Description	Arguments	Related Function
Utils.clearCookies()	Removes all cookies currently stored by the Vuser.		web_cleanup_cookies
Utils.clearCache()	Clears the contents of the cache simulator.		web_cache_cleanup

Ajax TruClient User Interface

This section includes (in alphabetical order):

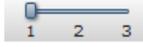
- ▶ Transaction Editor Dialog Box on page 540

Home Tab

This tab enables you to control the basic flow of the recording process for TruClient scripts.



User interface elements are described below:

UI Elements (A-Z)	Description
	Record. Starts recording the script. Additionally, you can use the arrow to specify whether to record before or after the selected step.
	Play. Replays the script. Additionally, you can use the arrow to specify whether to play the selected step only, or to run the script step by step. Running the script step by step pauses the replay after each step. For more information, see "Run The Script Step by Step" on page 518.
	Stop. Stops recording or replaying the script.
	Toggle Breakpoint. Toggles breakpoints on the selected step.
	Script Level. Modifies the script levels that are visible and replayed in the script. For more information, see "Script Levels" on page 506.
	Start/End Transaction. Inserts a starting or ending point for a transaction.
	Transaction Editor. Opens the Transaction Editor, allowing you to define new transactions and modify existing ones.
	Save. Saves the script.

Edit Tab

This tab enables you to cut, copy, and paste steps and data in TruClient scripts.



User interface elements are described below:

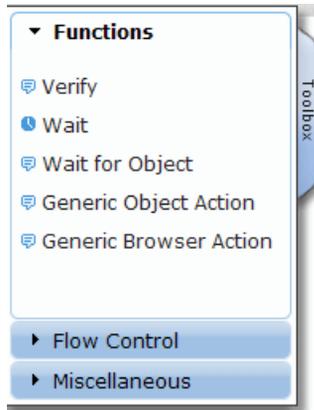
UI Elements (A-Z)	Description
	Cut the selected data or step.
	Copy the selected step or data.
	Pastes before the selected step.
	Pastes after the selected step.
	Pastes into the selected step.
	Deletes the selected step
	Opens the Find dialog box, allowing you to search the script for steps by step name or number.
	Go to the specified step.

Window Tab

This tab enables you to control multiple firefox windows for the same script. Select the window that contains the application that you want to bring to focus. This is needed for the debugging phase of script development, for example, when attempting to highlight a step object.

Toolbox

The toolbox enables you to add steps to TruClient scripts. The toolbox can be moved by dragging it up or down.

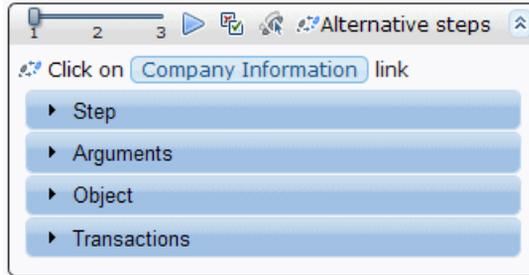


User interface elements are described below:

UI Elements (A-Z)	Description
Functions	<ul style="list-style-type: none"> ▶ Verify. Verify that an object exists in the application. ▶ Wait. Wait for a specified number of seconds before continuing with the next step. ▶ Wait for Object. Wait for an object to load before continuing with the next step. ▶ Generic Object/Browser Action. Blank steps that can be inserted and manually configured.
Flow Control	<ul style="list-style-type: none"> ▶ For Loop. A logical structure that repeats the steps contained in the loop a specified number of times. ▶ If Block. A logical structure that runs the steps contained in the block if the condition is met. ▶ Break. Causes the loop to end immediately without completing the current or remaining iterations. ▶ Continue. Causes the current loop iteration to end immediately. The script continues with the next iteration. ▶ Catch Error. Catches an error in the step immediately preceding and runs the contents of the catch error step. For more information, see "Insert Catch Error Steps" on page 517. ▶ Exit. Exits the iteration or the entire script depending on the specified setting.
Miscellaneous	<ul style="list-style-type: none"> ▶ Evaluate JavaScript. Runs the JavaScript code contained in the step. ▶ Evaluate JS on Object. Runs the JavaScript code contained in the step after the specified object is loaded in the application. ▶ Evaluate C. Runs the C code contained in the step. ▶ Comment. A blank step which allows you to write comments in your script.

TruClient Step Structure

TruClient steps are comprised of a number of sections. The sections and elements within each section vary depending of the type of step.



User interface elements are described below:

UI Elements (A-Z)	Description
	Script levels selector. Allows you to view and modify the script level of the step. For more information, see "Script Levels" on page 506.
	Replay. Replays this step only.
	Disable/Enable Step. Steps that are disabled are not replayed. This feature allows you to temporarily remove steps from the script without deleting them.
	Optional Step. Marking a step as optional means that in the event that the step can not find its object, the script continues without returning an error.
	Alternative Steps. This icon indicates a step which can be redefined in alternative ways. To redefine the step, click the icon, select the desired step definition, and click Back. For more information, see "Alternative Steps" on page 508.

UI Elements (A-Z)	Description
Step	<ul style="list-style-type: none"> ▶ Action: The action that defines the step. For steps with objects, this list is determined by the step roles. ▶ Object Timeout: If the object does not appear before this time in seconds, the step returns an error. ▶ Step Timeout: If the End Event is not reached by this time in seconds, the step returns an error. The way the script behaves when such an error occurs can be configured in the Run-Time settings dialog box. ▶ End Event: When the End Event occurs, the step allows the script to continue running subsequent steps.
Arguments	Contains step arguments. These arguments differ for different step actions and roles. For a list of the step arguments, see "TruClient Step Arguments" on page 526.

UI Elements (A-Z)	Description
Object	<ul style="list-style-type: none"> ▶ Roles: The functions that TruClient understands about an object. This information is read-only and is updated dynamically depending on how the object is used during recording. The list of available step actions is defined by these roles. ▶ Name: This is just a logical name and has no significance during replay. It can be modified to enhance readability. ▶ ID Method: The method of identifying the object. <ul style="list-style-type: none"> ▶ Automatic. TruClient's default object identification method. If this method does not successfully find the object during replay, click the Improve Object Identification button and replay the script again. ▶ XPath. Identifies the object based on an xpath expression that defines the object in the DOM tree. You can manually modify the expression. To regenerate the original expression generated by VuGen, click the Regenerate Expression button. ▶ JavaScript. JavaScript code that returns an object. For example: <code>document.getElementById("SearchButton")</code> returns an element that has a DOM ID attribute of "SearchButton".
Transactions	Allows you to create, modify, and view transactions. For more information, see "Insert Transactions" on page 516.

Transaction Editor Dialog Box

This dialog box enables you to manage transaction in Ajax TruClient vuser scripts.

UI Elements (A-Z)	Description
	Adds a new transaction or deletes the selected transaction.
General	Enables you to edit the name of the transaction
Start Point	The step and the event within the step which indicates the start of the transaction.
End Point	The step and the event within the step which indicates the end of the transaction.

TruClient Limitations

This section describes limitations for Ajax TruClient scripts.

- TruClient does not support the recording of Flash or Silverlight applications.
- TruClient support Mozilla Firefox applications, but should be able to record IE applications that are acid2 compliant.

14

AMF Protocol

This chapter includes:

Concepts

- ▶ AMF Protocol Overview on page 542
- ▶ AMF Terms on page 543
- ▶ AMF Example Script on page 544
- ▶ Setting the AMF Recording Mode on page 544

Concepts

AMF Protocol Overview

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.

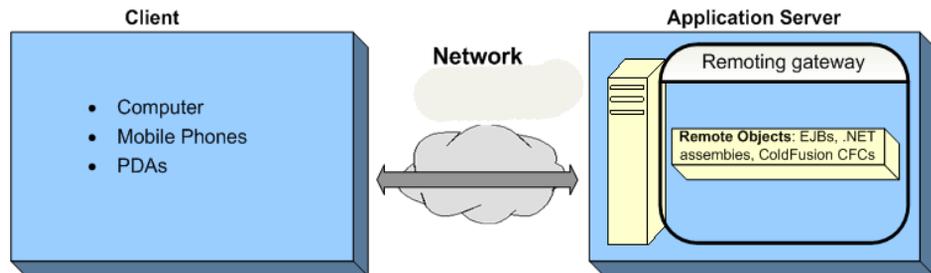
HTTP is supported by all Internet browsers and servers. Therefore, HTTP is a preferred method of communication between client applications and servers when working over the Internet.

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.



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 (Rich Internet Applications).
SOAP	A standard for exchanging XML-based messages over a computer network, normally using HTTP.

AMF Example Script

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**).

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 these options, select **Tools > Recording Options > General > Protocols** node. For more information, see "General Protocol Node" on page 348.

Note: If you record with one of the above options, you can modify the options and regenerate the script afterwards to include or exclude other protocols.

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,
    "Uri=movies/XMLExample.swf", "Referer=", ENDITEM,
    "Uri=movies/JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);

web_link("Sample JavaBean Movie Source",
    "Text=Sample JavaBean Movie Source",
    "Snapshot=t2.inf",
    EXTRARES,
    "Uri=XMLExample.swf", "Referer=", ENDITEM,
    "Uri=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\\x00ft\\x00\\x01\\x00/
flashgateway.samples.FlashJavaBean.testDocument\\x00\\x02/
1\\x00\\x00\\x004\\n\\x00\\x00\\x00\\x01\\x0F\\x00\\x00\\x00*<TEST
message=\"test\"><INSIDETEST /></TEST>",
    LAST);

```


15

Citrix Protocol

This chapter includes:

Concepts

- ▶ Citrix Protocol - Overview on page 552
- ▶ Citrix Recording Tips on page 553
- ▶ Citrix Replaying Tips on page 555
- ▶ Synchronization on page 557
- ▶ Automatic Synchronization on page 557
- ▶ Additional Synchronization on page 559
- ▶ Agent for Citrix Presentation Server Overview on page 561
- ▶ Troubleshooting Xenapp 5.0 on page 566

Tasks

- ▶ How to Configure the Citrix Client and Server on page 567
- ▶ How to Synchronize Citrix Scripts Manually on page 569
- ▶ How to Install and Uninstall the Citrix Agent on page 570

Reference

- ▶ Citrix Functions on page 572
- ▶ Understanding ICA Files on page 573
- ▶ Failed Bitmap Synchronization Dialog Box on page 574

Troubleshooting and Limitations on page 575

Concepts

Citrix Protocol - Overview

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 **ctx** prefix. These functions emulate the analog movements of the mouse and keyboard. In addition, the **ctx** functions allow you to synchronize the replay of the actions, by waiting for specific windows to open.

VuGen also allows you to record a Citrix NFUSE session. With Citrix NFUSE, the client is installed, but your interface is a browser instead of a client interface. To record NFUSE sessions, you must perform a multi-protocol recording for Citrix and Web Vusers. In multi-protocol mode, VuGen generates functions from both Citrix and Web protocols during recording.

In the following example, **ctx_mouse_click** simulates a mouse click on the left button.

```
ctx_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**).

Citrix 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.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Script Sections" on page 34.

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 "Automatic Synchronization" on page 557.

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**.

Citrix Replaying Tips

Wildcards

You can use wildcards (*) in defining window names. This is especially useful where the window name may change during replay, by its suffix or prefix.

In the following example, the title of the **Microsoft Internet Explorer** window was modified with a wildcard.

```
ctx_mouse_click(573, 61, LEFT_BUTTON, 0,
"Welcome to MSN.com - Microsoft Internet Explorer");
ctx_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 **ctx_sync_on_window** and **ctx_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 "How to Regenerate a Vuser Script" on page 111.

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.

Synchronization

Synchronization refers to waiting for windows and objects to become available before executing an action. This is necessary when recording Citrix scripts because, for example, if a step in a script opens a window, and the next step performs an action in that window, the second step cannot be implemented until the window opens. In order to ensure that VuGen does not replay the script incorrectly, it automatically generates functions that synchronize the script by waiting for windows or objects to become available. In addition, you can add synchronization functions manually.

For information about automatic synchronization, see "Automatic Synchronization" on page 557.

For information about manually adding synchronization points, see "How to Synchronize Citrix Scripts Manually" on page 569.

Automatic Synchronization

During recording, VuGen automatically generates steps that help synchronize the Vuser's replay of the script:

Sync on Window

The **Sync On Window** step instructs the Vuser to wait for a specific event before resuming replay. The available events are **Create** 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 `ctx_sync_on_window`.

Sync on Obj Info

The **Sync On Obj Info** step instructs the Vuser to wait for a specific object property before resuming replay. The available attributes are **Enabled**, **Visible**, **Focused**, **Text**, **Checked**, **Lines**, or **Item**. The Enabled, Visible, Focused, and Checked attributes are boolean values that can receive the values **true** or **false**. The other attributes require a textual or numerical object value.

A primary objective of this step is to wait for an object to be in focus before performing an action upon it.

VuGen automatically generates **sync_on_obj_info** steps when the Citrix agent is installed and the Use Citrix Agent Input in Code Generation option is enabled in the Recording options. By default, this Recording option is enabled. For more information, see "Citrix Code Generation Node" on page 327.

```
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 "Agent for Citrix Presentation Server Overview" on page 561), you can instruct VuGen to automatically generate a text synchronization step before each mouse click or double-click. By default, automatic text synchronization is disabled. For more information, see "Citrix Code Generation Node" on page 327.

Note, that even if you record a script with the option disabled, if you enable the option and regenerate the script, VuGen will insert text synchronization calls throughout the entire script.

In Script view, the corresponding function call to the **Sync On Text** step is **ctrx_sync_on_text_ex**.

The following segment shows a **ctx_sync_on_text_ex** function that was recorded during a Citrix recording with the HP Citrix Agent installed and text synchronization enabled.

```
ctx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,224,
"snapshot1", CTRX_LAST);
ctx_sync_on_text_ex (196, 198, 44, 14, "OK", "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);
ctx_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**).

Additional Synchronization

In addition, you can add several other steps that affect the synchronization indirectly:

Setting the Waiting Time

The **Set Waiting Time** step sets a waiting time for the other Citrix synchronization functions. This setting applies to all functions that follow it within the script. For example, if your **Sync on Window** steps are timing out, you can increase the default timeout of 60 seconds to 180.

To insert this step, select **Insert > Add Step > Set Waiting Time**.

Checking if a Window Exists or Closed

The `ctrx_win_exist` step checks if a window is visible in the Citrix client. By adding control flow statements, you can use this function to check for a window that does not always open, such as a warning dialog box. In the following example, `ctrx_win_exist` checks whether a browser was launched. The second argument indicates how long to wait for the browser window to open. If it did not open in the specified time, it double-clicks its icon.

```
if (!ctrx_win_exist("Welcome",6, CTRX_LAST))
    ctrx_mouse_double_click(34, 325, LEFT_BUTTON, 0, CTRX_LAST)
```

To insert this step, select **Insert > Add Step > Win Exist**.

Another useful application for this step is to check if a window has been closed. If you need to wait for a window to close, you should use a synchronization step such as **UnSet Window** or `ctrx_unset_window`.

For detailed information about these functions, see the *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 `ctx_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. Users 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.

```
ctx_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 Chapter 10, "Recording Options".

Agent for Citrix Presentation Server Overview

The Agent for Citrix Presentation Server, or Citrix Agent, is an optional utility that you can install on the Citrix server. It provides enhancements to the normal Citrix functionality. The following sections describe these enhancements.

It is provided in the product's installation disk and you can install it on any Citrix server. For more information about installing the Citrix Agent, see "How to Install and Uninstall the Citrix Agent" on page 570.

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);
```

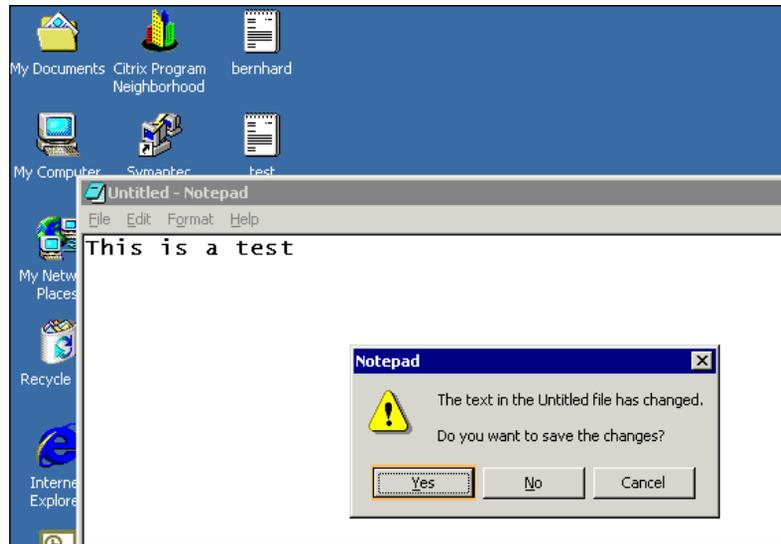
In the example above, the first argument of the `ctrx_obj_mouse_click` function contains the text of the window's title and the class, Notepad. Note that although the agent provides additional information about each object, Users only access objects by their window name and its coordinates.

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 `ctx_get_obj_info` function.
- ▶ **Insert Sync on Text and Get Text.** These are discussed in the section "Text Retrieval" on page 565.

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 `ctx_sync_on_obj_info` function provides synchronization by waiting for the Font dialog box to come into focus.

```
ctx_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:

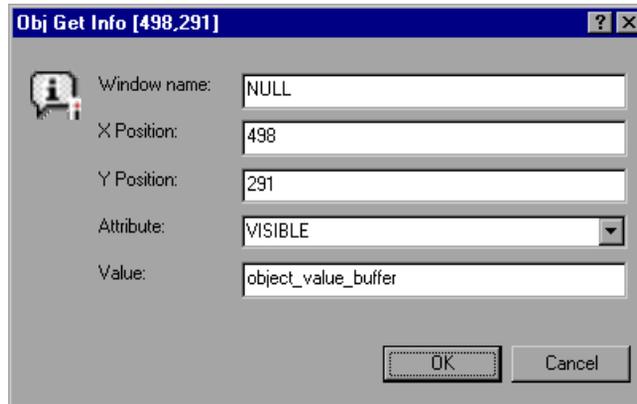
- 1** Click at a point within the tree view to insert the new step. Make sure that a snapshot is visible.
- 2** Click within the snapshot.
- 3** To mark a bitmap, right-click on it and select **Insert Sync on Bitmap**.

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.

For task details, see "How to Synchronize Citrix Scripts Manually" on page 569.

Troubleshooting Xenapp 5.0

When recording with Xenapp 5.0 using a Citrix and Web multi-protocol script, manual modifications are needed to ensure proper recording.

To record using Xenapp 5.0

1 Modify XeApp Settings:

- a** Connect to the XenApp server through a web interface.
- b** Select **Preferences > Session Settings** and set **Window Size** to **No Preference**.

2 Update Correlation Rules

3 Modify Recording Options

- a** Open the **General > Recording** node of the Recording Options dialog box.
- b** Select **Advanced HTML** to open the Advanced HTML dialog box.
- c** Select **A script containing explicit URL's only**.

Tasks

How to Configure the Citrix Client and Server

Before creating a script, make sure you have a supported Citrix client installed on your machine, and that your server is properly configured. The following steps describe this process.

- "Configure the Citrix client" on page 567
- "Install the Metaframe server" on page 568
- "Configure the Metaframe server" on page 568

Configure the Citrix client

In order to run your script, you must install a Citrix client on each Load Generator machine. If you do not have a client installed, you can download one from the Citrix Website www.citrix.com under the **download** section.

VuGen supports all Citrix clients with the exception of versions 8.00, version 6.30.1060 and earlier, and Citrix Web clients.

Install the Metaframe server

Make sure the MetaFrame server (3, 4, or 4.5) is installed. To check the version of the server, select **Citrix Connection Configuration** on the server's console toolbar and select **Help > About**.

Configure the Metaframe server

Configure the Citrix server to completely close a session. After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time that client opens a new connection. Consequently, a new connection by the same client will face the same workspace from which it disconnected previously. It is preferable to allow each new test run to use a clean workspace.

To 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**.

How to Synchronize Citrix Scripts Manually

In addition to the automatic synchronization, you can manually add synchronization both during and after recording. A common use of this capability is where the actual window did not change, but an object within the window changed. Since the window did not change, VuGen did not detect or record a **Sync on Window**.

For example, if you want the replay to wait for a specific graphic image in a browser window, you insert manual synchronization. Or, if you are recording a large window with several tabs, you can insert a synchronization step to wait for the new tab's content to open.

Synchronize manually during recording

To add synchronization during recording, you use the floating toolbar. The **Sync On Bitmap** and **Sync on Text** functions lets you to mark an area or text within the client window that needs to be in focus before resuming replay.



- To insert a **Sync on Bitmap** step, click the **Insert Sync on Bitmap** button on the toolbar and mark a rectangle around the desired area.



- To insert a **Sync on Text** step (Citrix Agent required), click the **Insert Sync On Text** button on the toolbar and mark a rectangle around the desired text.

Synchronize manually after recording

You can also add synchronization after the recording session. To add a synchronization step, right-click in the snapshot 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)

How to Install and Uninstall the Citrix Agent

The installation file for the Agent for Citrix Presentation Server is located on the LoadRunner installation disk, under the **Additional Components\ Agent for Citrix Presentation Server** folder.

Note that the agent should only be installed on your Citrix server machine—not Load Generator machines.

Install the Agent for Citrix Presentation Server

- 1 If you are upgrading the agent, make sure to uninstall the previous version before installing the next one.
- 2 If your server requires administrator permissions to install software, log in as an administrator to the server.
- 3 If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

- 4 Locate the installation file, **Setup.exe**, on the product installation disk in the **Additional Components\ Agent for Citrix Presentation Server\Win32 or Win64** directory.
- 5 Follow the installation wizard to completion.

Note: After installation the agent will only be active for LoadRunner invoked Citrix sessions—it will not be active for users who start a Citrix session without LoadRunner.

Uninstall the Agent for Citrix Presentation Server

- 1 If your server requires administrator privileges to remove software, log in as an administrator to the server.

- 2** Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Citrix Presentation Server 32 or 64** and click **Change/Remove**.

Reference

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 **ctx** prefix. For example, **ctx_obj_mouse_click** emulates a mouse click for a specific object. You can manually edit or add any of the functions into your Vuser script after the recording session.

For more information about the **ctx** 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.

Understanding ICA Files

Citrix ICA client files are text files that contain configuration information for the applications accessed through the Citrix client. These files must have an .ica extension and must conform to the following format:

```
[WFClient]
Version=
TcpBrowserAddress=

[ApplicationServers]
AppName1=

[AppName1]
Address=
InitialProgram=#
ClientAudio=
AudioBandwidthLimit=
Compress=
DesiredHRES=
DesiredVRES=
DesiredColor=
TransportDriver=
WinStationDriver=

Username=
Domain=
ClearPassword=
```

Note: When you load an ICA file using the Recording Options, VuGen saves the file together with your script, eliminating the need to copy the ICA file to each load generator machine.

The following example shows a sample ICA file for using Microsoft Word on a remote machine through the Citrix client:

```
[WFClient]
Version=2
TcpBrowserAddress=235.119.93.56

[ApplicationServers]
Word=

[Word]
Address=Word
InitialProgram=#Word
ClientAudio=On
AudioBandwidthLimit=2
Compress=On
DesiredHRES=800
DesiredVRES=600
DesiredColor=2
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

Username=test
Domain=user_lab
ClearPassword=test
```

For more information, see the Citrix Website www.citrix.com.

Failed Bitmap Synchronization Dialog Box

This dialog box enables you to decide what to do when a bitmap synchronization fails.

To access	When there is a mismatch between the Recording and Replay snapshots in a bitmap synchronization step, this dialog box opens automatically during replay.
------------------	--

User interface elements are described below:

UI Elements (A-Z)	Description
Continue	Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.
Recording Snapshot	A view of the recording snapshot.
Replay Snapshot	A view of the replay snapshot.
Stop	Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution by default. Alternatively, you can specify Continue on Error for a specific function as described in "Continuing on Error" on page 576.

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Citrix Protocol.

Effects and Memory Requirements of Citrix Agent

When you run Citrix Vusers with the agent installed, each Vuser runs its own process of **ctxagent.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 **ctxagent.exe** process) is approximately 4.35 MB. To run 25 Vusers, you would need 110 MBs of memory.

Data Execution Prevention (DEP) and Citrix Agent Performance

DEP is a security feature included in Windows version XP Service Pack 2 and later. DEP can interfere with some functions of the Agent for Citrix Presentation Server and may cause VuGen to stop recording.

If you experience unusual behavior during recording in these environments, modify the DEP settings.

To modify the Windows DEP settings:

- 1** Open **Start > Control Panel > System**.
- 2** In the Advanced tab, click **Performance settings**.
- 3** In the Performance Options Data Execution Prevention tab, select the first option, **DEP for essential services only**.

If you cannot change this option, click **Add**. Browse to the client program, for example IEXPLORE.EXE.

If neither of these options are possible, try to disable DEP completely.

- a** In the Control Panel, click the **Advanced** tab under **System** section.
 - b** Under **Startup and Recovery**, click **Edit**.
 - c** Replace **NoExecute=OptOut** with **NoExecute=AlwaysOff**.
- 4** Click **OK** to save the settings.
 - 5** Reboot the machine.

Debugging Tips

The following section lists debugging tips for Citrix scripts.

Single Client Installation

If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine. To verify that only one client is installed, open the Add/Remove Programs dialog box from the Control Panel and make sure that there is only one entry for the Citrix ICA client.

Add Breakpoints

Add breakpoints to your script in VuGen to help you determine the problematic lines of code.

Synchronize Your Script

If replay fails, you may need to insert synchronization functions into your script to allow more time for the desired windows to come into focus. Although you can manually add a delay using `lr_think_time`, we recommend that you use one of the synchronization functions discussed in "Automatic Synchronization" on page 557.

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.

Recording with XenApp 5.0

When recording with Xenapp 5.0 using a Citrix and Web multi-protocol script, manual modifications are needed to ensure proper recording.

To record using XenApp 5.0

1 Modify XenApp Settings

- a** Connect to the XenApp server through Internet Explorer.
- b** Log in using the same account that will be used during recording.

- c** Select **Preferences > Session Settings** and set **Window Size** to **No Preference**.

2 Update Correlation Rules

- a** Open the **HTTP Correlation** node of the Recording Options dialog box.
- b** Make sure that the **Citrix_XenApp** rule is selected.

3 Modify Recording Options

- a** Open the **General > Recording** node of the Recording Options dialog box.
- b** Select **Advanced HTML** to open the Advanced HTML dialog box.
- c** Select **A script containing explicit URL's only**.

16

Click and Script Protocols

This chapter includes:

Concepts

- ▶ Recording Tips on page 582
- ▶ Replay Tips on page 583
- ▶ Miscellaneous Tips on page 585
- ▶ Web (Click and Script) Enhancements on page 586

Reference

- ▶ Web (Click and Script) API Notes on page 592

Troubleshooting and Limitations on page 593

Concepts

Recording Tips

Use the mouse and not the keyboard

It is preferable to click on an object with the mouse rather than using the keyboard. During recording, use only GUI objects that are within the browser's pane. Do not use any browser icons, controls, the Stop button, or menu items, such as **View > Refresh**. You may, however, use the Refresh, Home, Back and Forward buttons and the address bar.

Do not record over an existing script

It is best to record into a newly created script—not an existing one.

Avoid context menus

Avoid using context menus during recording. Context menus are 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 594.

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. You can enable the **Recording Options > GUI Properties > Advanced** node.

Replay Tips

The following section lists tips for replaying click and script scripts.

Do not reorder

Do not change the order of the statements within a recorded script. Also, copying segments of code from one Action to another is not recommended.

Convert non-ASCII characters

If your links contain non-ASCII characters, you should instruct VuGen to convert the data to or from the UTF-8 format.

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 is 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 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 path name of krb5.ini into the KRB5_CONFIG environment variable.

The krb5.ini file should contain detailed information about each domain (KDS and AS addresses) and trust chains.

For more information, contact HP software support.

Web (Click and Script) Enhancements

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 User 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 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`.

Example script (Web Click and Script)

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);...
```

Reference

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);
```

Troubleshooting and Limitations

This section describes troubleshooting and limitations for Click and Script protocols. Some items apply to specific click and script protocols only.

Recording Issues and Limitations

Firefox is not supported

Only Internet Explorer is supported for Web (Click 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 583).

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. These settings can be found in the **Recording Options > GUI Properties > Web Event Configuration** node.

Certain user actions were not recorded

Check if there is a Java applet running inside the browser. If not, record the script with the Web (HTTP/HTML) protocol.

Replay Issues

GUI object not found

Does the error occur at the beginning of the second iteration?

If the error occurs at the beginning of the second iteration's Action section, it is probably the result of a starting page that was present for the first iteration, but missing for the second one. If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page, so that the same link will be visible at the end of the business process.

Is it a text link containing non-ASCII characters?

If the problem occurs with non-ASCII characters, you should instruct VuGen to convert 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 failure

If the replay is failing at a particular step, check the step description. VuGen sometimes reads a single space as a double space. Make sure that there are no incorrect double spaces in the string.

Miscellaneous Issues

Out of memory error in JavaScript

Increase the JavaScript memory in the Run-Time settings.

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.

17

COM Protocol

This chapter includes:

Concepts

- COM Protocol Overview on page 600
- COM Technology Overview on page 600
- COM Script Structure on page 602
- COM Sample Scripts on page 604
- Selecting COM Objects to Record on page 610

Concepts

COM Protocol Overview

When you record COM client applications, VuGen generates functions that describe COM client-server activity. The recorded script contains interface declarations, API calls and instance calls to methods. Each COM function begins with an `Irc` prefix. You can configure the programming language in which to create a Vuser script as either C or Visual Basic.

For each COM Vuser script, VuGen creates the following:

- ▶ Interface pointer and other variable declarations in 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

COM Technology Overview

This section provides an outline of COM technology. This should be enough to get you started with COM Vuser scripts. See Microsoft Developer's Network (MSDN) and other documentation for further details.

COM (Component Object Model) is a technology for developing reusable software components ("plug-ins"). DCOM (Distributed COM) allows use of COM components on remote computers. Microsoft transaction servers (MTS), Visual Basic and Explorer all use COM/DCOM technology. Thus, the application you are testing may use COM technology indirectly, even though you don't know it. You will probably have to include some, but certainly not all, of the COM calls made by your application in the Vuser script.

Objects, Interfaces and Type Libraries

COM objects are binary code modules. Each COM object implements one or more interfaces that allow client programs to communicate with it. You need to know about these interfaces in order to follow the COM calls in the Vuser scripts. Type libraries, used as a reference for accessing COM interface methods and parameters, contain descriptions of COM objects and interfaces. Each COM class, interface, and type library is identified by a Global Unique Identifier (GUID).

COM Interfaces

A COM interface provides a grouped collection of related methods. For example, a **Clock** object may have **Clock**, **Alarm** and **Timer** interfaces. Each interface has one or more methods. For example the **Alarm** interface may have **AlarmOn** and **AlarmOff** methods.

An interface may also have one or more properties. Sometimes, the same function may be performed by calling a method or by setting or getting the value of a property. For example, you can set the **Alarm Status** property to **On** or call the **AlarmOn** method.

A COM object may support many interfaces. The **IUnknown** interface is implemented by all components and is used to find out about other interfaces. Many components also implement the **IDispatch** interface, which exposes all other interfaces and methods of the object, allowing implementation of COM automation in scripting languages.

COM Class Context and Location Transparency

COM objects can run on the same machine as the client application, or on a remote server. COM objects that an application creates may be in a local library, a local process or a remote machine ("Remote Object Proxy"). The location of the COM object, known as the "Context," can be transparent to the application. Most users apply the Vusers to check the load on remote servers. Therefore, objects accessed by Remote Object Proxy are usually the most relevant for these tests.

COM Data Types

COM also provides several special data types, including safe arrays, BSTR strings and variants. You may need to use these data types for debugging, parameterization and similar tasks.

COM 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 = Irc_GUID("student.student.1");
IUnknown * pUnkOuter = (IUnknown*)NULL;
unsigned long dwClsContext = Irc_ulong("7");
GUID riid = IID_IUnknown;
Irc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid, (void*)&IUnknown_0,
CHECK_HRES);
}
```

Error Checking

Each COM method or API call returns an error value. VuGen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

CHECK_HRES	This value is inserted if the function passed during recording and errors should be checked during replay.
DONT_CHECK_HRES	This value is inserted if the function failed during recording and errors should not be checked during replay.

COM Sample Scripts

This section shows examples of how VuGen emulates a COM client application. It is divided up into the basic COM script operations. Each type of operation is done within a separate scope.

Instantiation of the Object

To use a COM object, the application must first instantiate it and get a pointer to an interface of that object.

VuGen does the following to instantiate an object:

- 1 VuGen calls `Irc_GUID` to get a unique ProgID for the object, to be stored in `pClsid`:

```
GUID pClsid = Irc_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 = Irc_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 `Irc_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:

```
Irc_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:

```
Irc_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 **Irc_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 `Irc_DispatchMethod` passes optional arguments "#3" and "var3":

```
{
    GUID riid = IID_IDispatch;
    Irc_IOptional_QueryInterface(IOptional_0, &riid, (void*)&IOptional_0,
    CHECK_HRES);
}
{
    VARIANT P1 = Irc_variant_short("47");
    VARIANT P2 = Irc_variant_short("37");
    VARIANT P3 = Irc_variant_date("3/19/1901");
    VARIANT var3 = Irc_variant_scode("4");
    Irc_DispatchMethod((IDispatch*)IOptional_0, "in_out_optional_args", /*locale*/1024,
    &P1, &P2, OPTIONAL_ARGS, "#3", &P3, "var3", &var3, LAST_ARG, CHECK_HRES);
}
```

The different `Irc_Dispatch` 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 `Irc_DispatchMethod1` call was used to retrieve an array of name strings:

```
VARIANT retValue = Irc_variant_empty();
retValue = Irc_DispatchMethod1((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 = Irc_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**)

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.

18

Database Protocols

This chapter includes:

Concepts

- ▶ Database Protocols Overview on page 614
- ▶ VuGen Database Recording Technology on page 615
- ▶ Database Grids on page 616
- ▶ Oracle Applications on page 618
- ▶ Handling Errors on page 619
- ▶ Debugging Database Applications on page 622

Reference

Troubleshooting and Limitations on page 624

Concepts

Database Protocols Overview

Suppose that you have a database of customer information that is accessed by customer service personnel located throughout the country. You use Database Vusers to emulate the situation in which the database server services many requests for information. A Database Vuser could:

- ▶ connect to the server
- ▶ submit an SQL query
- ▶ retrieve and process the information
- ▶ disconnect from the server

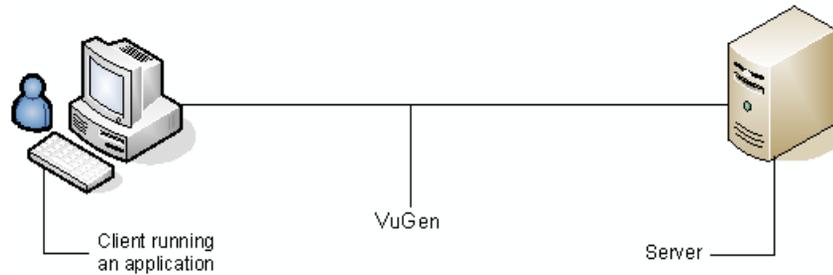
You distribute several hundred Database Vusers among the available load generators, each Vuser accessing the database by using the server API. This enables you to measure the performance of your server under the load of many users.

The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. The Vusers execute the script and emulate user load on the client/server system. The Vusers generate performance data which you can analyze in report and graph format.

VuGen supports the following database types: CtLib, DbLib, Informix, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity.

VuGen Database Recording Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.



Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.



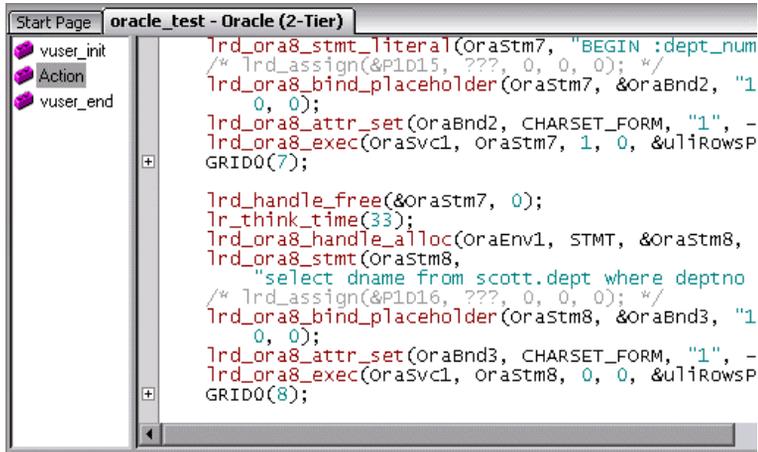
You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and UNIX environments.

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 Chapter 46, "Creating and Running Script in UNIX."

Database 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.



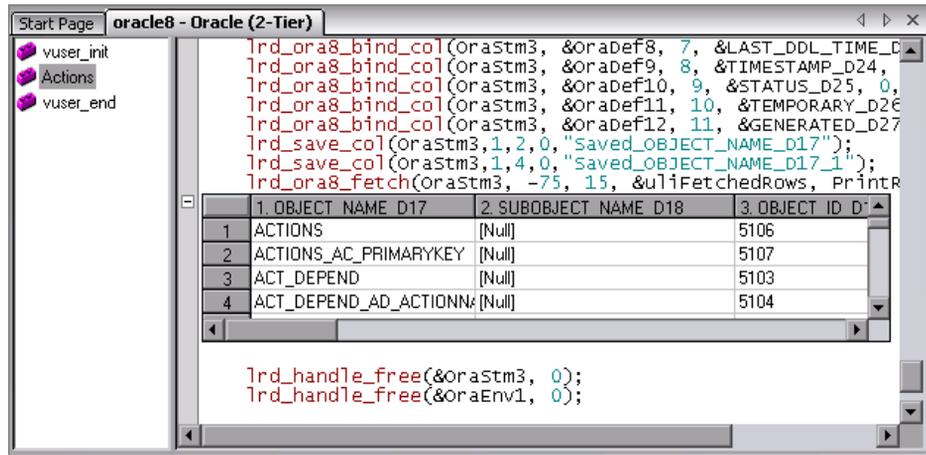
```

lrd_ora8_stmt_literal(OraStm7, "BEGIN :dept_num
/* lrd_assign(&P1D15, ???, 0, 0, 0); */
lrd_ora8_bind_placeholder(OraStm7, &oraBnd2, "1
0, 0);
lrd_ora8_attr_set(OraBnd2, CHARSET_FORM, "1", -
lrd_ora8_exec(OraSvc1, OraStm7, 1, 0, &ul1RowsP
GRID0(7);

lrd_handle_free(&oraStm7, 0);
lr_think_time(33);
lrd_ora8_handle_alloc(OraEnv1, STMT, &oraStm8,
lrd_ora8_stmt(OraStm8,
"select dname from scott.dept where deptno
/* lrd_assign(&P1D16, ???, 0, 0, 0); */
lrd_ora8_bind_placeholder(OraStm8, &oraBnd3, "1
0, 0);
lrd_ora8_attr_set(OraBnd3, CHARSET_FORM, "1", -
lrd_ora8_exec(OraSvc1, OraStm8, 0, 0, &ul1RowsP
GRID0(8);

```

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.



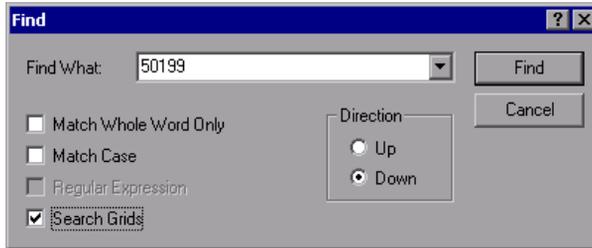
If the data value is very long, only part of it is shown in the grid. This truncation only occurs in the displayed grid and has no impact on the data.

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:

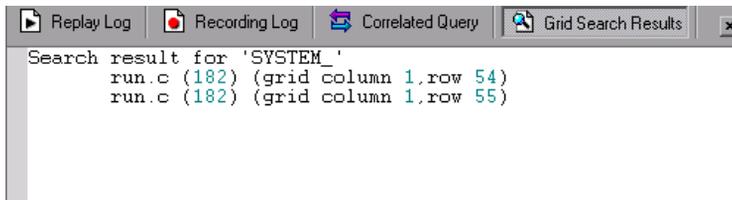
```
[general]
max_line_at_grid=200
```

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.



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.

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 different ways as described below.

Globally Modifying Error Handling

You can change the way that Vusers handle errors by issuing an `LRD_ON_ERROR_CONTINUE` or `LRD_ON_ERROR_EXIT` statement. By default, a Vuser aborts the script execution when it encounters any type of error—database, parameter related, and so on. To change the default behavior, insert the following line into your script:

```
LRD_ON_ERROR_CONTINUE;
```

From this point on, the Vuser continues script execution, even when an error occurs.

You can also specify that the Vuser continue script execution when an error occurs only within a segment of the script. For example, the following code tells the Vuser to continue script execution even if an error occurs in the `lrd_stmt` or `lrd_exec` functions:

```
LRD_ON_ERROR_CONTINUE;
lrd_stmt(Csr1, "select...");
lrd_exec(...);
LRD_ON_ERROR_EXIT;
```

Use the `LRD_ON_ERROR_CONTINUE` statement with caution, as significant and severe errors may be missed.

Locally Modifying Error Handling

You can set error handling for a specific function by modifying the severity level. Functions such as `lrd_stmt` and `lrd_exec`, which perform database operations, use severity levels. The severity level is indicated by the function's final parameter, `miDBErrorSeverity`. This parameter tells the Vuser whether or not to continue script execution when a database error occurs (error code 2009). The default, 0, indicates that the Vuser should abort the script when an error occurs.

For example, if you reference a table that does not exist, the following database statement fails and the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
1 /*Dflt Ora Ver*/, 0);
```

To tell a Vuser to continue script execution, even when a database operation error occurs for that function, change the statement's severity parameter from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
1 /*Dflt Ora Ver*/, 1);
```

When the severity is set to 1 and a database error occurs, a warning is issued. Note that the severity level set for a particular function applies only to that function.

CtLib Result Set Errors

In CtLib recording, the application retrieves all of the available result sets after executing a statement. If the returned result set contains fetchable data, the application performs bind and fetch operations on the data as indicated in the following example:

```
lrd_stmt(Csr15, "select * from all_types", -1, 148, -99999, 0);
lrd_exec(Csr15, 0, 0, 0, 0, 0);
lrd_result_set(Csr15, 1 /*Succeed*/, 4040 /*Row*/, 0);
lrd_bind_col(Csr15, 1, &tinyint_D41, 0, 0);
...
lrd_fetch(Csr15, -9, 0, 0, PrintRow3, 0);
```

If a result set does not contain fetchable data, bind and fetch operations cannot be performed.

When you parameterize your script, result data may become unfetchable (depending on the parameters). Therefore, a CtLib session that recorded bind and fetch operations for a particular statement, may not be able to run, if the new data is unfetchable. If you try to execute an **lrd_bind_col** or an **lrd_fetch** operation, an error will occur (LRDRET_E_NO_FETCHABLE_DATA — error code 2064) and the Vuser will terminate the script execution.

You can override the error by telling the Vuser to continue script execution when this type of error occurs. Insert the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_CONT;
```

To return to the default mode of terminating the script execution, type the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_EXIT;
```

Use this option with caution, as significant and severe errors may be missed.

Debugging Database Applications

The following tips apply to database applications only (Oracle, ODBC, Ctlib)

Generating Debugging Information

Note: You can now set options to view most of the information described in this section using VuGen's user interface.

VuGen contains an inspector "engine." You can force VuGen recorder to create "inspector" output by editing `\WINDOWS_DIR\vugen.ini` as follows:

```
[LogMode]
EnableAscii=ASCII_LOG_ON
```

When this option is enabled, VuGen creates a file, **vuser.asc** in the Data 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 Ird 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.

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for database protocols.

Troubleshooting all database protocols

IE crashes when recording Oracle NCA/11i scripts

This can occur due to an incompatible dll file.

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);
```

Two-tier Database Scripting Tips

The following section offers solutions for two-tier database scripts.

Question 1: Why does the script fail when it is data driven, while the same values work with the application itself?

Answer: The failure may be a result of trailing spaces in your data values. Even though the data values that you type directly into the GUI are probably truncated, you should manually eliminate them from your data file. Tab-delimited files can hide trailing spaces and therefore obscure problems. In general, comma-delimited files are recommended. You can view the files in Excel to see if things are correct.

Question 2: Why does an SQL error of an invalid cursor state occur on the second iteration?

Answer: The `lrd_close_cursor` function may not have been generated or it may be in the *end* section instead of the *action* section. You will need to add a cursor close function or move it from the *end* section to make the script iterate successfully.

Opening a new cursor may be costly in terms of resources. Therefore, we recommend that you only open a cursor once in the *actions* section during the first iteration. You can then add a new parameter that contains the iteration number as a string by using the Iteration Number type. Call this parameter *IterationNum*. Then, inside the *actions* section replace a call to open a new cursor 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, 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.

In 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:

```
"lrd.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960:
ambiguous column naming in select list".
```

Answer: Change the statement by adding an alias to at least one of the non-unique columns, thereby mapping it to a new unique name. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE second, DESCRIPTION
FROM"
"MTL_UNITS_OF_MEASURE "
"WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
"SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

Troubleshooting Oracle 2-Tier Users

This section contains a list of common problems that you may encounter while working with Oracle Users, 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: lrd_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

19

Enterprise Java Beans (EJB) Protocol

This chapter includes:

Concepts

- EJB Testing Overview on page 638
- EJB Detector Output and Log Files on page 639

Tasks

- How to Create an EJB Vuser Script on page 640
- How to Run an EJB Vuser Script on page 644
- How to Install and Run the EJB Detector on page 645
- How to Set and Verify the Java Environment on page 649

Reference

- EJB User Interface on page 652
- EJB Vuser Script Examples on page 653

Concepts

EJB Testing Overview

VuGen provides several tools for developing a script that tests Java applications. For generating a Vuser script through recording, use the Java Record and Replay Vuser. For creating a script through programming, use the Custom Java Vuser type.

EJB Testing Vusers differ from the standard Java Vusers in that VuGen automatically creates a script to test or tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. VuGen's EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to test or tune, and VuGen generates a script that emulates each of the EJB's methods.

It creates transactions for each method so that you can measure its performance and locate problems. In addition, each method is wrapped in a **try and catch** block for exception handling.

Note that in order to create EJB testing scripts, the EJB Detector must be installed and active on the application server host. The Detector is described in the following sections.

VuGen also has a built-in utility for inserting methods into your script. Using this utility, you display all of the available packages, select the desired methods, and insert them into your script. For more information, see "How to Run an EJB Vuser Script" on page 644.

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.

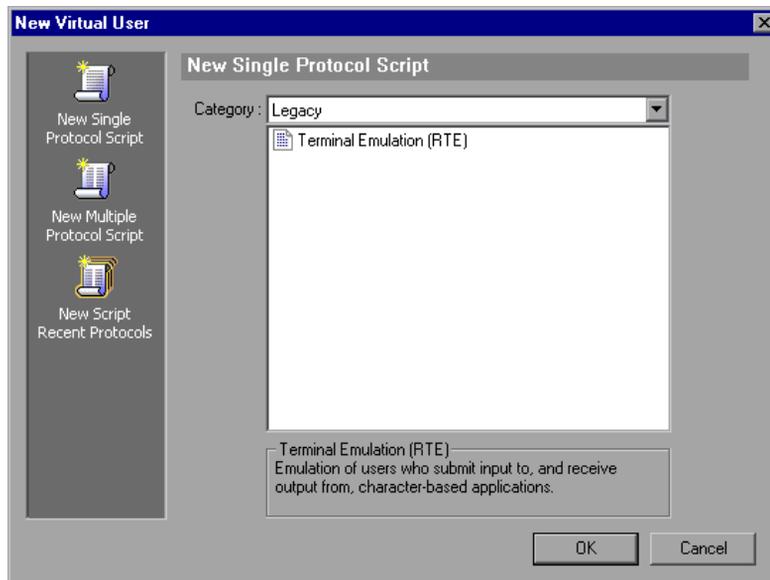
Tasks

How to Create an EJB Vuser Script

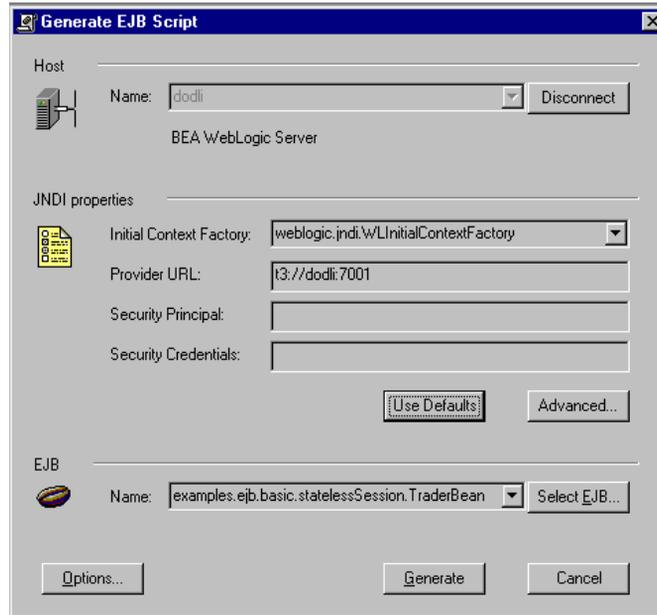
Creating an EJB Vuser Script differs from creating Vuser Scripts using other protocols. This task describes how to create an EJB Vuser Script.

To create an EJB Vuser script:

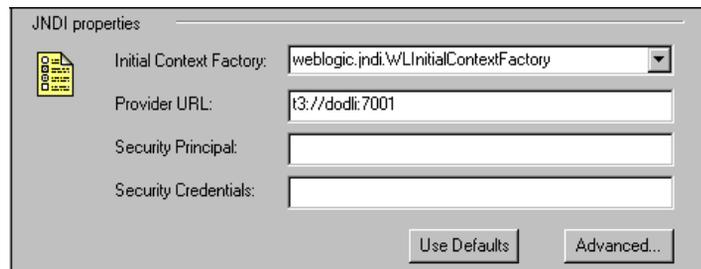
- 1 Select **File > New** or click the **New** button. The New Virtual User dialog box opens.



- 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.



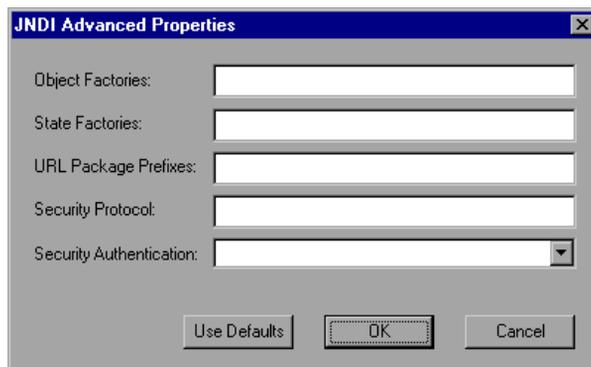
- 4** The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory** and the **Provider URL**.

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:

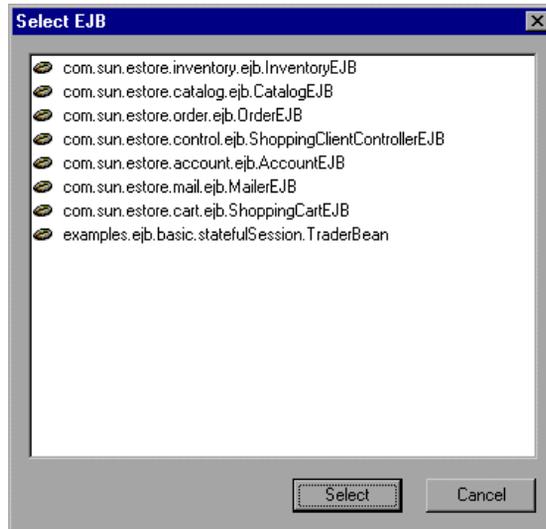
Type	Initial Context Factory	Provider URL
WebLogic	weblogic.jndi.WLInitialContextFactory	t3://<appserver_host>:7001
WebSphere 3.x	com.ibm.ejs.ns.jndi.CNInitialContextFactory	iiop://<appserver_host>:900
WebSphere 4.x	com.ibm.websphere.naming.WsnInitialContextFactory	iiop://<appserver_host>:900
Sun J2EE	com.sun.enterprise.naming.SerialInitContextFactory	N/A
Oracle	com.evermind.server.ApplicationClientInitialContextFactory	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.

How to Run an EJB Vuser Script

After you generate a script for your EJB testing, and make the necessary modifications, you are ready to run your script. The EJB script allows you to perform two types of testing: functional and load. The functional testing verifies that the EJB, functions properly within your environment. The load testing allows you to evaluate the performance of the EJB when executing many users at one time.

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 657).
- 3** Insert additional methods, and modify their default values. For more information, see the section on inserting Java functions in Chapter 21, "Java Protocol."
- 4** Configure the Run-Time Settings. For more information, see "Run-Time Settings" on page 417.
- 5** Make sure you have a valid Java environment. For more information, see "How to Set and Verify the Java Environment" on page 649
- 6** 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.

How to Install and Run the EJB Detector

This task describes how to install and run the EJB Detector. The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

This task includes the following steps:

- ▶ "Install the EJB Detector" on page 645
- ▶ "Run the EJB Detector" on page 645

1 Install the EJB Detector

- a Verify that you have a valid JDK environment on the machine.
- b Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
- c Unzip the `<LR_root>\ejbcomponent\ejbdetector.jar` file into the EJB Detector directory.

2 Run the EJB Detector

The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

- ▶ To run the EJB Detector from the command line, see "How to Run the EJB Detector from the command line" on page 646.
- ▶ To run the EJB Detector from a batch file, see "How to Run the EJB Detector from a batch file" on page 648.

How to Run the EJB Detector from the command line

This task describes how to run the EJB Detector from the command line

To run the EJB Detector from the command line:

- 1** Add the DETECTOR_HOME\classes and the DETECTOR_HOME\classes\xerces.jar to the CLASSPATH environment variable.
- 2** If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the CLASSPATH:
- 3** For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar
- 4** For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar
- 5** If your EJBs use additional classes directory or .jar files, add them to the CLASSPATH.

6 Enter the following string in the command line:

```
java EJBDetector [search root dir] [listen port]
```

search root dir	<p>One or more directories or files in which to search for EJBs (separated by semicolons). Follow these guidelines:</p> <p>BEA WebLogic Servers 4.x and 5.x. Specify the application server root directory.</p> <p>BEA WebLogic Servers 6.x. Specify full path of the domain folder.</p> <p>WebSphere Servers 3.x. Specify the full path of the deployed EJBs folder.</p> <p>WebSphere Servers 4.0. Specify the application server root directory.</p> <p>Oracle OC4J. Specify the application server root directory.</p> <p>Sun J2EE Server. Specify the full path to the deployable .ear file or directory containing a number of .ear files.</p> <p>If unspecified, the classpath will be searched.</p>
listen port	<p>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.</p> <p>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.</p>

How to Run the EJB Detector from a batch file

You can launch the EJB detector using a batch file, `EJB_Detector.cmd`. This file resides in the root directory of the EJB Detector installation, after you unzip `ejbdetector.jar`.

To run the EJB Detector from a batch file:

- 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 <code>.ear</code> file or directory containing a number of <code>.ear</code> files.
EJB_DIR_LIST (optional)	List of directories/files, separated by ';' and containing deployable <code>.ear/.jar</code> files, and any additional classes directory or <code>.jar</code> 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).

How to Set and Verify the Java Environment

This task describes the methods of setting and verifying the java environment.

This task includes the following steps:

- "For Websphere 3.x users:" on page 649
- "For WebSphere 4.0 users:" on page 650
- "For Oracle OC4J users:" on page 650
- "For Sun J2EE users:" on page 651
- "For WebLogic 4.x - 5.x Users:" on page 651
- "For WebLogic 6.x and 7.0 users:" on page 651

1 For Websphere 3.x users:

- a** Using the IBM JDK 1.2 or higher:
 - Add the `<WS>\lib\ujc.jar` to the classpath.
- b** 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.

2 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.

3 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.

4 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.

5 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.

6 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.

Reference

EJB User Interface

This section includes:

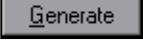
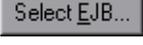
- ▶ Generate EJB Script Dialog Box on page 652

Generate EJB Script Dialog Box

Enables you to begin recording an EJB script.

To Access	File > New > Enterprise Java Beans (EJB)
Relevant Tasks	"How to Install and Run the EJB Detector" on page 645

User interface elements are described below:

UI Element (A-Z)	Description
	Opens the JNDI Advanced Properties dialog box. Specify the desired properties.
	Opens relevant recording options dialog box. For more information see "Recording Options" on page 307.
 	Connects to and disconnects from the server. You must be connect to the server to generate a script.
	Generates the Vuser script.
	Opens the Select EJB dialog box.
EJB Name	The name of EJB for which you want to create a test.

UI Element (A-Z)	Description
Host Name	The machine on which the EJB Detector is installed. The detector must be running in order to connect.
JNDI Properties	The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the Initial Context Factory and the Provider URL .

EJB Vuser Script Examples

VuGen generates a script that tests your EJB, based on the JNDI (Java Naming and Directory Interface) properties you specified when creating the Vuser script. JNDI is Sun's programming interface used for connecting Java programs to naming and 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();
        }
    }
}
```

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://bealocig:7001")
```

You can specify the provider URL before recording, so you don't have to modify it manually, in the **JNDI 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.

20

Flex Protocol

This chapter includes:

Concepts

- ▶ Flex Overview on page 660
- ▶ Code Generation Notifications on page 661

Tasks

- ▶ How to Work with Flex RTMP on page 662
- ▶ How to Query an XML Tree on page 662
- ▶ How to Handling Externalizable Objects in Flex on page 665

Reference

- ▶ Flex Functions and Example on page 668

Troubleshooting and Limitations on page 670

Concepts

Flex Overview

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** protocol 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 protocol.

Code Generation Notifications

If a Flex, Silverlight, or Java over HTTP script encounters errors during the code generation phase, the **Code Generation Notifications** dialog box automatically opens. This dialog box displays details about each error, as well as recommended actions. Follow the recommended actions and regenerate the script.

If the error is related to externalizable objects in a Flex script, follow the procedure "How to Handling Externalizable Objects in Flex" on page 665.

To manually open this dialog box at any time, select **Tools > Generation Errors**.

Tasks

How to Work 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.

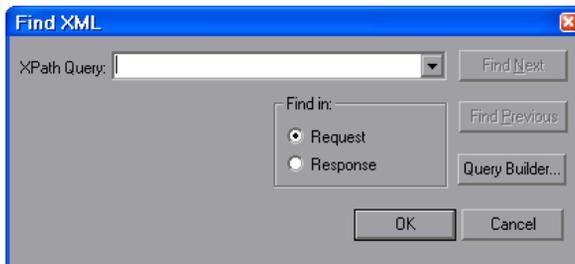
How to Query an XML Tree

VuGen provides a Query Builder that lets you create and execute queries on the XML.

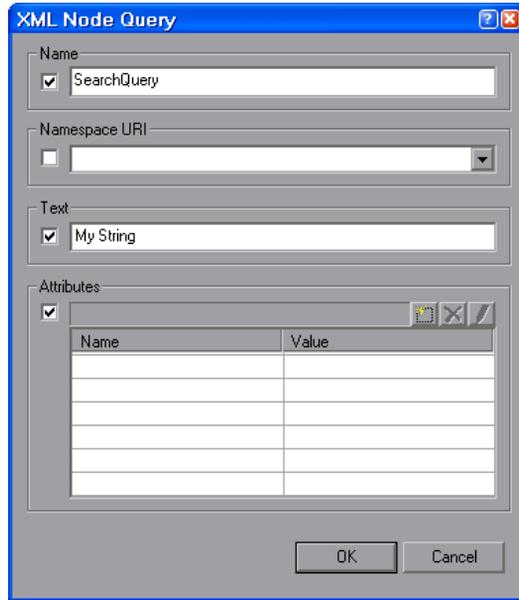
VuGen displays the XML code in an expandable tree. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

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**.



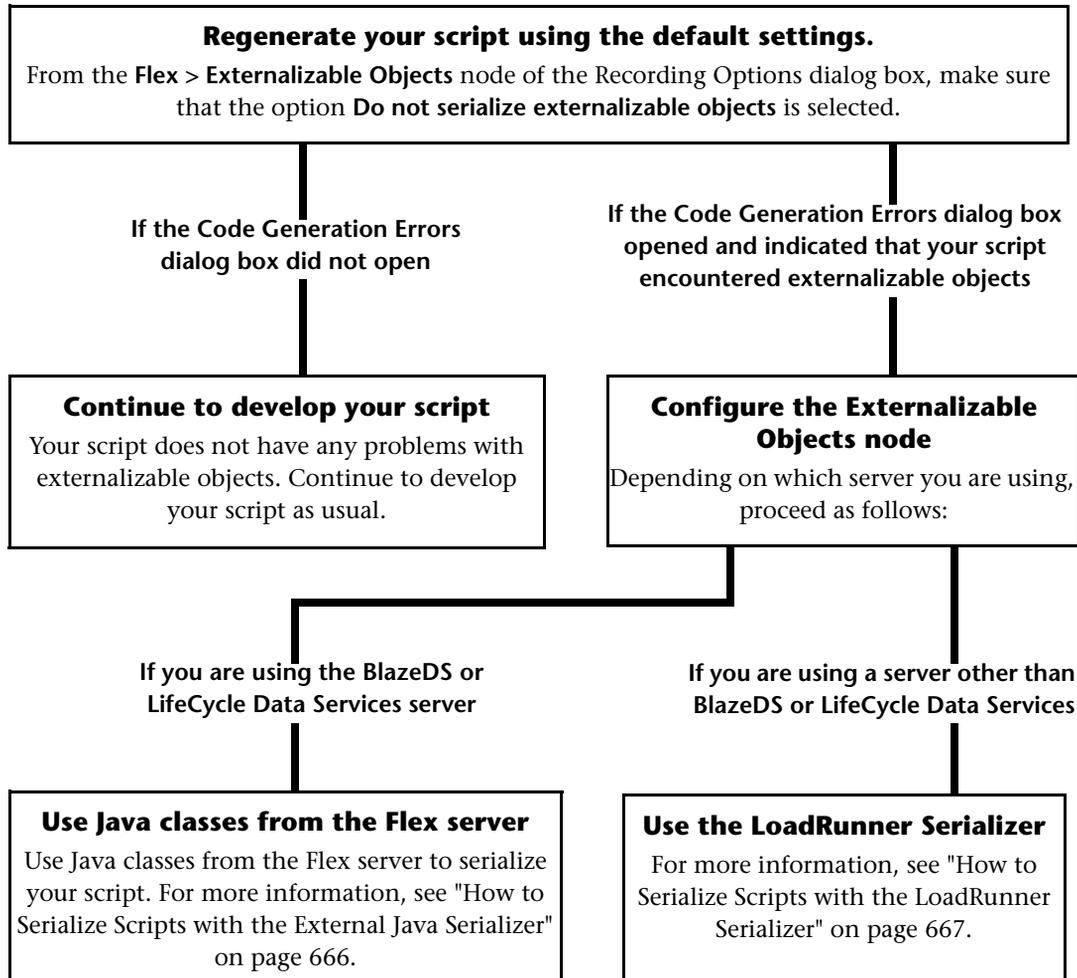
- 9 Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the Find XML box.



- 10 Click **Find Next** to begin the search.

How to Handling Externalizable Objects in Flex

This task graphically describes the workflow for dealing with externalizable objects in Flex scripts. You configure the settings for working with externalizable objects in the **Flex > Externalizable Objects** node of the Recording Options dialog box.



Note: You can leave externalizable objects in binary form (unserialized) by selecting **Do not serialize externalizable objects**. This method of recording allows you to replay the script as recorded. However correlating and working with binary data can be very difficult.

How to Serialize Scripts with the External Java Serializer

This task describes how to use the Java classes from the Flex server to serialize AMF messages in your script.

To handle externalizable objects using an external serializer

- 1** In the **Recording Options > Flex > Externalizable Objects** node, select **Serialize objects using** and select **Custom Java Classes** from the drop-down menu.
- 2** Add the relevant files by using the **Add all classes in folder** or **Add JAR or Zip file** buttons. We recommend adding files incrementally as follows:
 - a** Depending on the server type, add the following JAR files:
 - ▶ **BlazeDS:** flex-messaging-common.jar, flex-messaging-core.jar
 - ▶ **LCDS:** flex-messaging.jar, flex-messaging-common.jar
 - b** Regenerate the script and note any errors that mention missing java classes.
 - c** Open the recording options dialog box using the **Generation Options** button and add the necessary files.
 - d** Regenerate the script and repeat the procedure until there are no more missing Java classes.
 - e** If you receive any other errors follow the recommendations in the Code Generation Errors dialog box.
- 3** Ensure that the added files exist in the same location both on the VuGen machine and on all load generators.



Notes and Limitations for the Java Serializer

- Supported JDK versions: 1.5 and earlier.
- Supported servers: BlazeDS and LifecycleDS.
- Microsoft .NET classes are not supported.
- Serialization/Deserialization methods must be implemented in the same way for both the server and client side.
- Externalizable Java classes must implement the default constructor.
- VuGen use the Java classes out of the server run-time context, therefore any code that relies on the context may result in errors.
- During code generation VuGen performs a validity test of the request buffers by verifying that the buffer can be read and written using the provided jars. Failure in this validity test (for example, a null pointer exception) indicates that the classes are incompatible with LoadRunner.



How to Serialize Scripts with the LoadRunner Serializer

You can attempt to serialize externalizable objects using the LoadRunner serializer. Ensure that you have saved all open scripts because this option may result in unexpected errors or invalid steps.

To use the LoadRunner serializer:

- 1** Save all open scripts in VuGen.
- 2** In the **Recording Options > Flex > Externalizable Objects** node, select **Serialize objects using** and select **LoadRunner AMF serializer** from the drop-down menu.

Reference

Flex Functions and Example

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_envelope_header_set	Defines a set of envelope headers.

Example:

In the following example, `flex_ping` checks for the availability of a service. The `flex_remoting_call` function invokes the service remotely.

```
flex_ping("1",
  "URL=http://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**).

Troubleshooting and Limitations

This section describes troubleshooting and limitations for Flex protocol scripts.

Problems with Binary Data in Flex Steps

When recording a Flex application, information is usually passed between the client and server using known serialization methods (AMF). However, when a given AMF object uses a custom serialization method (externalizable), VuGen generates a **flex_web_request** step containing unserialized binary data instead of a **flex_amf_call** step. When this occurs, the **Code Generation Errors** dialog box opens automatically after code generation with a message indicating that there was a problem with externalizable objects.

For information about how to resolve this issue, see "How to Handling Externalizable Objects in Flex" on page 665.

21

Java Protocol

This chapter includes:

Concepts

- ▶ Java Protocol Recording Overview on page 673
- ▶ Java Vuser Script Overview on page 674
- ▶ RMI over IIOP Overview on page 675
- ▶ Corba Recording Options on page 676
- ▶ CORBA Application Vendor Classes on page 676
- ▶ Recording RMI on page 677
- ▶ Recording a Jacada Vuser on page 677
- ▶ Working with CORBA on page 678
- ▶ Working with RMI on page 680
- ▶ Working with Jacada on page 681
- ▶ Java Custom Filters - Overview on page 683
- ▶ Java Custom Filters - Determining which Elements to Include on page 684

Tasks

- ▶ How to Record a Java Vuser Script on page 686
- ▶ How to Record Java Scripts Using Windows XP and 2000 Server on page 687
- ▶ How to Run a Script as Part of a Package on page 688
- ▶ How to Manually Insert Java Methods on page 688

- ▶ How to Manually Configure Script Generation Settings on page 690
- ▶ How to Create a Custom Java Filter on page 694

Reference

- ▶ Hook File Structure on page 696
- ▶ Java Icon Reference List on page 699

Concepts

Java Protocol Recording Overview

Using VuGen, you can record a Java application or applet. VuGen creates a pure Java script enhanced with Vuser API Java-specific functions. After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it. Once you verify that the script is functional, you incorporate it into a LoadRunner scenario or Business Process Monitor configuration.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. See Chapter 22, "Java Protocol - Manually Programming 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 "How to Convert Web Vuser Scripts into Java" on page 897.

After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it.

You integrate finished scripts into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Service Management* documentation.

Java Vuser Script Overview

When you record a session, VuGen logs all calls to the server and generates a script with functions. These functions describe all of your actions within the application or applet. The script also contains supplementary code required for proper playback, such as property settings, and naming service initialization (JNDI).

The recorded script is comprised of three sections:

- Imports
- Code
- Variables

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script. The **Code** section contains the Actions class and the recorded code within the **init**, **actions**, and **end** methods. The **Variables** section, after the **end** method, contains all the type declarations for the variables used in the code.

After you finish recording, you can modify the functions in your script, or add additional Java or LoadRunner functions to enhance the script. Note that if you intend to run Java Vusers as threads, the Java code you add to your script must be thread-safe. For details about function syntax, see the *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 706.

RMI over IIOP Overview

The **Internet Inter-ORB Protocol (IIOP)** technology was developed to allow implementation of CORBA solutions over the World Wide Web. IIOP lets browsers and servers exchange complex objects such as arrays, unlike HTTP, which only supports transmission of text.

RMI over IIOP technology makes it possible for a single client to access services which were only accessible from either RMI or CORBA clients in the past. This technology is a hybrid of the JRMP protocol used with RMI and IIOP used with CORBA. **RMI over IIOP** allows CORBA clients to access new technologies such as **Enterprise Java Beans (EJB)** among other J2EE standards.

VuGen provides full support for recording and replaying Vusers using the **RMI over IIOP** protocol. Depending on what you are recording, you can utilize VuGen's RMI recorder to create a script that will optimally emulate a real user:

- ▶ **Pure RMI client.** recording a client that uses native JRMP protocol for remote invocations
- ▶ **RMI over IIOP client.** recording a client application that was compiled using the IIOP protocol instead of JRMP (for compatibility with CORBA servers).

Corba Recording Options

For recording a CORBA session, you need to set the following options in the Recording Options:

- JNDI
- Use DLL hooking to attach VuGen support

CORBA Application Vendor Classes

Running CORBA applications with JDK1.2 or later, might load the JDK internal CORBA classes instead of the specific vendor CORBA classes. To force the virtual machine to use the vendor classes, specify the following java.exe command-line parameters:

Visigenic 3.4

```
-Dorg.omg.CORBA.ORBClass=com.visigenic.vbroker.orb.ORB  
-Dorg.omg.CORBA.ORBSingletonClass=com.visigenic.vbroker.orb.  
  ORBSingleton
```

Visigenic 4.0

```
-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB  
-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton
```

OrbixWeb 3.x

```
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB  
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.  
  singletonORB
```

OrbixWeb 2000

```
-Dorg.omg.CORBA.ORBClass=com.ion.corba.art.artimpl.ORBImpl  
-Dorg.omg.CORBA.ORBSingletonClass=com.ion.corba.art.artimpl.  
  ORBSingleton
```

Recording RMI

Before recording an RMI session, verify that your application or applet functions properly on the recording machine.

Before you record, verify that your environment is configured properly. Make sure that the required classes are in the classpath and that you have a full installation of JDK. For more information on the required environment settings, see "Set the Java environment" on page 711

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 34, "Web 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.

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 Irapi.Ir;

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 Ir.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 Ir.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 (Ir.get_vuser_id() == -1)
        orb.shutdown();

    return Ir.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 "Correlating Java Scripts - Serialization" on page 192.

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 22, "Java Protocol - Manually Programming Scripts."

Java Custom Filters - Overview

This section describes the background information necessary to create custom Java filters. For task details, see "How to Create a Custom Java Filter" on page 694.

When testing your Java application, your goal is to determine how the server reacts to client requests. When load testing, you want to see how the server responds to a load of many users. With VuGen's Java Vuser, you create a script that emulates a client communicating with your server.

VuGen provides filter files that define hooking properties for commonly used methods. There are filter definitions for RMI, CORBA, JMS, and JACADA protocols. You can also define custom filters.

When you record a method, the methods which are called from the recorded method either directly or indirectly, will not be recorded.

In order to record a method, VuGen must recognize the object upon which the method is invoked, along with the method's arguments. VuGen recognizes an object if it is returned by another recorded method provided that:

- the construction method of that object is hooked
- it is a primitive or a built-in object
- It supports a serializable interface.

You can create a custom filter to exclude unwanted methods. When recording a Java application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters for RMI, CORBA, JMS, and JACADA protocols were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your JAVA application's calls or exclude unnecessary calls. Custom JAVA protocols, proprietary enhancements and extensions to the default protocols, and data abstraction all require a custom filter definition.

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, VuGen allows you to record with a stack trace that logs all of the methods that were called by your application. In order to record with stack trace set the log level to **Detailed**.

Java Custom Filters - Determining which Elements to Include

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- ▶ **Top Down Approach.** An approach in which you include the relevant package and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined layer which implements all client-server activity without involving any GUI elements.
- ▶ **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT packages and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- ▶ If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- ▶ If you identify a non-client/server call in your script, exclude its method in the filter.

- During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen serializes it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by deserializing it.
- VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the `lr.deserialize()` method in your script. For more information see "Correlating Java Scripts - Serialization" on page 192.
- Exclude all activity which involves GUI elements.
- Add classes for utilities that may be required for the script to be compiled.

Tasks

How to Record a Java Vuser Script

This task describes how to record a Java vuser script.

This task includes the following steps:

- "Prerequisites" on page 687
- "Create a new Java Record Replay Protocol script" on page 687
- "Complete the Start Recording dialog box" on page 687
- "Set the Recording Options" on page 687

1 Prerequisites

Make sure that you have properly installed a JDK version from Sun on the machine running the Vusers—JRE alone is insufficient. Verify that the classpath and path environment variables are set according to the JDK installation instructions. Before you replay a Vuser script, verify that your environment is configured properly for the JDK and relevant Java classes.

2 Create a new Java Record Replay Protocol script

Select **File > New** and select **JAVA Record Replay** from the **Java** category.

3 Complete the Start Recording dialog box

Enter the details of your application in the Start Recording dialog box. For user interface details, see "Start Recording Dialog Box" on page 119.

4 Set the Recording Options

In the Start Recording dialog box, click **Options** to open the Recording Options dialog box. In the **Recorder Options** node, the **Recorded Protocol** field configures the main protocol you will be recording. If you are recording more than one Java protocol, enter the additional protocols in the **Extension List** field.

How to Record Java Scripts Using Windows XP and 2000 Server

When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

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. Clear the **Microsoft Internet Explorer** check box and click **Apply**. Then, reselect the **Microsoft Internet Explorer** check box and click **Apply**.

How to Run a Script as Part of a Package

This section is not relevant for Jacada type scripts.

When creating or recording a Java script, you may need to use methods from classes in which the method or class is protected. When attempting to compile such a script, you receive compilation errors indicating that the methods are not accessible.

To use the protected methods, add the `Vuser` to the package of required methods. At the beginning of your script, add the following line:

```
package a.b.c;
```

where **a.b.c** represents a 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.

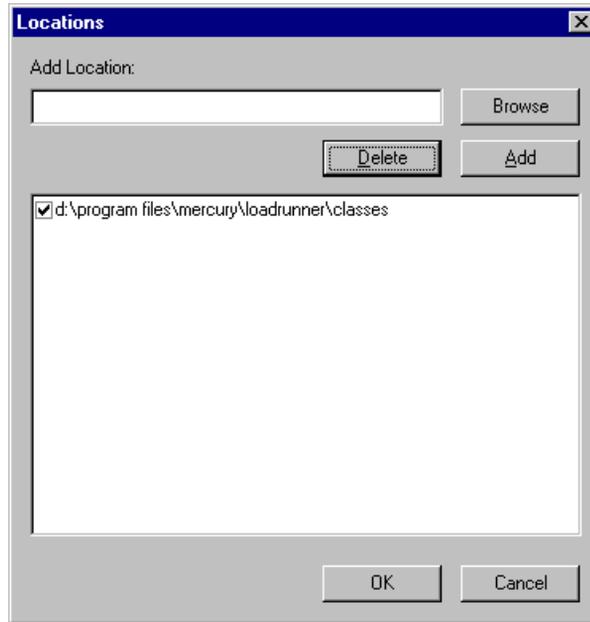
How to Manually Insert Java Methods

You use the Java Function navigator to view and add Java functions to your script. You can customize the function generation settings by modifying the configuration file. For more information, see "General Script Node" on page 353.

To insert Java functions:

- 1 Click within your script at the desired point of insertion.
- 2 Select **Insert > Insert Java Function**. The Insert Java Function dialog box opens. The lower part of the dialog box displays a description of the Java object.

- 3** Click **Locations**. The Locations dialog box opens. By default, VuGen lists the paths defined in the CLASSPATH environment variable.



- 4** Click **Browse** to add another path or archive to the list. To add a path, select **Browse > Folder**. To add an archive (**jar** or **zip**), select **Browse > File**. When you select a folder or a file, VuGen inserts it in the **Add Location** box.
- 5** Click **Add** to add the item to the list.
- 6** Repeat steps 4 and 5 for each path or archive you want to add.
- 7** Select or clear the check boxes to the left of each item in the list. If an item is checked, its members will be listed in the Java Class navigator.
- 8** Click **OK** to close the Locations dialog box and view the available packages.
- 9** Click the plus and minus signs to the left of each item in the navigator, to expand or collapse the trees.
- 10** Select an object and click **Paste**. VuGen places the object at the location of the cursor in the script. To paste all the methods of a class into your script, select the class and click **Paste**.

- 11 Repeat the previous step for all of the desired methods or classes.
- 12 Modify the parameters of the methods. If the script generation setting **DefaultValues** is set to **true**, you can use the default values inserted by VuGen. If **DefaultValues** is set to **false**, you must add parameters for all methods you insert into the script.

In addition, modify any return values. For example, if your script generated the following statement "(String)=LavaVersion.getVersionId();", replace (String) with a string type variable.

- 13 Add any necessary statements to your script such as imports or Vuser API Java functions (described in Chapter 22, "Java Protocol - Manually Programming Scripts").
- 14 Save the script and run it from VuGen.

How to Manually Configure Script Generation Settings

You can customize the way the navigator adds methods to your script.

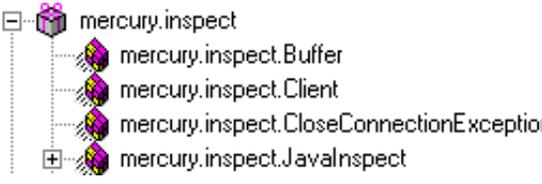
To view the configuration setting, open the **jquery.ini** file in VuGen's dat directory.

```
[Display]
FullClassName=False

[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

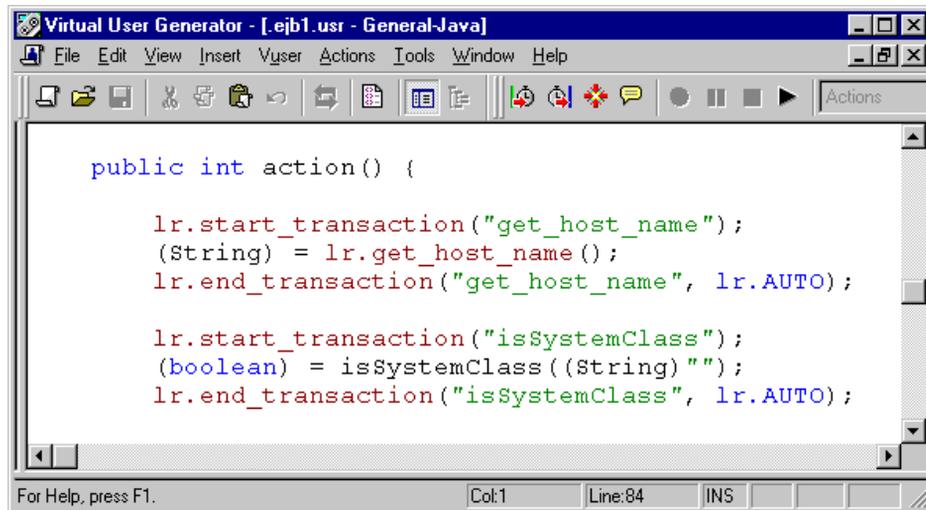
Class Name Path

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.



```

public int action() {

    lr.start_transaction("get_host_name");
    (String) = lr.get_host_name();
    lr.end_transaction("get_host_name", lr.AUTO);

    lr.start_transaction("isSystemClass");
    (boolean) = isSystemClass((String) "");
    lr.end_transaction("isSystemClass", lr.AUTO);
}

```

The screenshot shows the Virtual User Generator (VuGen) interface. The title bar reads "Virtual User Generator - [.ejb1.usr - General-Java]". The menu bar includes File, Edit, View, Insert, Vuser, Actions, Tools, Window, and Help. The toolbar contains various icons for file operations, execution, and debugging. The main text area displays a Java script snippet with transaction wrappers. The status bar at the bottom indicates "For Help, press F1.", "Col:1", "Line:84", and "INS".

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
<code>lr.message((String) "");</code>	<code>lr.message((String));</code>
<code>lr.think_time((int)0);</code>	<code>lr.think_time((int));</code>
<code>lr.enable_redirection((boolean>false);</code>	<code>lr.enable_redirection((boolean));</code>
<code>lr.save_data((byte[])null, (String) "");</code>	<code>lr.save_data((byte[]), (String));</code>

Class Pasting

The **CleanClassPaste** setting pastes a class so that it will compile cleanly: with an instance returning from the constructor, with default values as parameters, and without a need for import statements. Using this option, you will most likely be able to run your script without any further modifications. If you disable this option (default), you may need to manually define parameters and include import statements. Note that this setting is only effective when you paste an entire class into your script—not when you paste a single method.

The following segment shows the `toString` method pasted into the script with the `CleanClassPaste` option enabled.

```
_class.toString();
// Returns: java.lang.String
```

The same method with the `CleanClassPaste` option disabled is pasted as follows:

```
(String) = toString();
```

The next segment shows the **NumInserter** Constructor method pasted into the script with the `CleanClassPaste` option enabled.

```
utils.NumInserter _numinserter = new utils.NumInserter
    ((java.lang.String)"", (java.lang.String)"", (java.lang.String)""...);
// Returns: void
```

The same method with the `CleanClassPaste` option disabled is pasted as:

```
new utils.NumInserter((String)"", (String)"", (String)""...);
```

How to Create a Custom Java Filter

This task describes how to create a custom Java filter. For background information, see "Java Custom Filters - Overview" on page 683.

For details of the Hook File structure, see "Hook File Structure" on page 696.

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.

To define an custom java filter:

- 1** Create a new filter based on one of the built-in filters by modifying the **user.hooks** file which is located in the product's **classes** directory. For structural details about the user.hook file, see "Hook File Structure" on page 696.
- 2** Open the Recording Options (Ctrl+F7) and select the **Log Options** node. Select the Log Level to **Detailed**.
- 3** Record your application. Click **Start Record** (Ctrl + R) to begin and **Stop** (Ctrl + F5) to end.
- 4** View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain and correlate, you should customize the script's filter.
- 5** Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) or by viewing a Stack Trace of the script.

- 6 Set the filter to include the relevant methods. For more information, see "Java Custom Filters - Determining which Elements to Include" on page 684.
- 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 "Correlating Java Scripts" on page 189 and "Correlating Java Scripts - Serialization" on page 192.

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.

Reference

Hook File Structure

The following section describes the structure of a typical .hooks file:

```
[Hook-Name]
class    = MyPackage.MyClass
method  = MyMethod
signature = ()V
ignore_cl =
ignore_mtd =
ignore_tree =
cb_class = mercury.ProtocolSupport
cb_mtd =
general_cb = true
deep_mode = soft | hard
make_methods_public = true | false
lock = true | false
```

The hook files are structured as .ini files where each section represents a hook definition. Regular expressions are supported in some of the entries. Any entry that uses regular expression must start with a '!'.

Hook-Name

Specifies the name of this section in the hooks file. Hook-Name must be unique across all hooks files. A good practice is to give the fully qualified class name and method. For example:

```
[javax.jms.Queue.getQueueName]
```

Class

A fully qualified class name. Regular expression can be used to include several classes from the same package, a whole package, several packages, or any class that matches a name. For example:

```
Class = !javax\.jms\.*
```

Method

The simple name of the method to include. Regular expressions can be used to include more than one method from the class. For example:

```
Method = getQueueName
```

Signature

The standard Java internal type signature of the method. To determine the signature of a method, run the command `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.

Java Icon Reference List

The following table describes the icons that represent the various Java objects:

Icon	Item	Example
	Package	<code>java.util</code>
	Class	<code>public class Hashtable extends java.util.Dictionary implements java.lang.Cloneable, java.io.Serializable</code>
	Interface Class (gray icon)	<code>public interface Enumeration</code>
	Method	<code>public synchronized java.util.Enumeration keys ()</code>

	Static Method (yellow icon)	public static synchronized java.util.TimeZone getTimeZone
	Constructor Method	public void Hashtable ()

22

Java Protocol - Manually Programming Scripts

This chapter includes:

Concepts

- ▶ Manually Programming Java Scripts - Overview on page 702
- ▶ Java Protocol Programming Tips on page 703
- ▶ Running Java Vuser Scripts on page 705
- ▶ Compiling and Running a Script as Part of a Package on page 706

Tasks

- ▶ How to Manually Create a Java Script on page 707
- ▶ How to Enhance a Java Script on page 711

Concepts

Manually Programming Java Scripts - Overview

To prepare Vuser scripts using Java code, use the **Java** type Vusers. This Vuser type supports Java on a protocol level. The Vuser script is compiled by a Java compiler and supports all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes `"/` `/"`.

Chapter 21, "Java Protocol" explains how to create a script through recording using the **Java Record Replay** Vuser. To prepare a Java coded script through programming, see the following sections.

The first step in creating a Java compatible Vuser script, is to create a new Vuser script template of the type **Java Vuser**. Then, you program or paste the desired Java code into the script template. You can add Java Vuser functions to enhance the script and parameterize the arguments to use different values during iterations.

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

After you prepare a script, run it as a standalone test from VuGen. A Java compiler (Sun's `javac`), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the *HP LoadRunner Controller, Performance Center, or HP Business Service Management* documentation.

Java Protocol Programming Tips

When programming a Java Vuser script, you can paste ready-made code segments into scripts or import ready-made classes in order to invoke their methods. If Vusers need to run as threads under the Controller (for scalability reasons), you need to make sure that all of the imported code is thread-safe.

Thread-safety is often difficult to detect. A Java Vuser may run flawlessly under VuGen and under the Controller with a limited number of Vusers. 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 Irapi.*;
public class Actions
{
    private static int iteration_counter = 0;

    public int init() {
        return 0;
    }

    public int action() {
        iteration_counter++;
        return 0;
    }

    public int end() {
        Ir.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 11, "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;
}
```

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 Irapi.*;
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).

Tasks

How to Manually Create a Java Script

This task describes how to manually create and edit a custom Java script.

This task includes the following steps:

- "Create a new script" on page 707
- "Insert your code into the script" on page 708
- "Insert additional LoadRunner API functions" on page 710
- "Insert additional Java functions" on page 710

1 Create a new script

- a** Open VuGen.
- b** Select **File > New** or click the **New** button. The New Virtual User dialog box opens.
- c** Select **Custom > Java Vuser** from the Select Vuser type list, and click **OK**. VuGen displays a blank Java Vuser script.
- d** Click the **Actions** section in the left frame to display the **Actions** class.

2 Insert your code into the script

After generating an empty template, you can insert the desired Java code. When working with this type of Vuser script, you place all your code in the Actions class. To view the Actions class, click **Actions** in the left pane. VuGen displays its contents in the right pane.

```
import Irapl.*;
public class Actions
{
    public int init() {
        return 0;
    }

    public int action() {
        return 0;
    }

    public int end() {
        return 0;
    }
}
```

The Actions class contains three methods: init, action, and end. The following table shows what to include in each method and when each method is executed.

Script method	Used to emulate...	Is executed when...
init	a login to a server	the Vuser is initialized (loaded)
action	client activity	the Vuser is in "Running" status
end	a log off procedure	the Vuser finishes or is stopped

Init Method

Place all the login procedures and one-time configuration settings in the `init` method. The `init` method is only executed once—when the Vuser begins running the script. The following sample `init` method initializes an applet. Make sure to import the `org.omg.CORBA.ORB` function into this section, so that it will not be repeated for each iteration.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import Irapl.Ir;

// 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 iVuser 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 11, "Run-Time Settings." The following sample action method retrieves and prints the Vuser ID.

```
public int action() {
    Ir.message("vuser: " + Ir.get_vuser_id() + " xxx");
    return 0;
}
```

End Method

In the **end** method, place the code you want the Vuser to execute at the end of the script, such as logging off from a server, cleaning up the environment, and so forth.

The end method is only executed once—when the Vuser finishes running the script. In the following example, the end method closes and prints the end message to the execution log.

```
public int end() {  
    lr.message("End");  
    return 0;  
}
```

3 Insert additional LoadRunner API functions

VuGen provides a specific Java API for Java Vuser scripts. These functions are all static methods of the `lrapi.lr` class.

The Java API functions are classified into several categories: Transaction, Command Line Parsing, Informational, String, Message, and Run-Time functions.

For further information about each of these functions, see the *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.

4 Insert additional Java functions

To use additional Java classes, import them at the beginning of the script as shown below.

Remember to add the classes 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  
{  
    ...  
}
```

5 Add script enhancements

You add script enhancements such as rendezvous points, transactions, and output messages. For more information, see "How to Enhance a Java Script" on page 711.

6 Set the Java environment

Before running your Java Vuser script, make sure that the environment variables, `PATH` and `CLASSPATH`, are properly set on all machines running Vusers:

- To compile and replay the scripts, you must have complete JDK installation, either version 1.1 or 1.2, or 1.3. The installation of the JRE alone is not sufficient. It is preferable not to have more than one JDK or JRE installation on a machine. If possible, uninstall all unnecessary versions.
- The `PATH` environment variable must contain an entry for `JDK/bin`.
- For JDK 1.1.x, the `CLASSPATH` environment variable must include the `classes.zip` path, (`JDK/lib` 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.

How to Enhance a Java Script

This task describes how to enhance custom Java scripts.

Inserting Transactions

You define transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified requests. These requests can be short or complex tasks. When working with LoadRunner, you can analyze the performance per transaction during and after the scenario run, using online monitor and graphs.

You can also specify a transaction status: `lr.PASS` or `lr.FAIL`. You can let the Vuser automatically determine if the transaction was successful, or you can incorporate it into a conditional loop. For example, in your code you can check for a specific return code. If the code is correct, you issue a `lr.PASS` status. If the code is wrong, you issue an `lr.FAIL` status.

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 Service Management.

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 11, "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 the following format, where the arguments and their values are listed in pairs after the script name:

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

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

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

For more information on the command line parsing functions, see the *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 Service Management* documentation.

23

Java over HTTP Protocol

This chapter includes:

Concepts

- ▶ Java over HTTP Protocol Overview on page 720
- ▶ Viewing Responses and Requests in XML Format on page 720

Tasks

- ▶ How to Record with Java over HTTP on page 722
- ▶ How to Debug Java over HTTP scripts on page 724
- ▶ How to Insert Parameters into Java over HTTP Scripts on page 725

Reference

Troubleshooting and Limitations on page 726

Concepts

Java over HTTP Protocol Overview

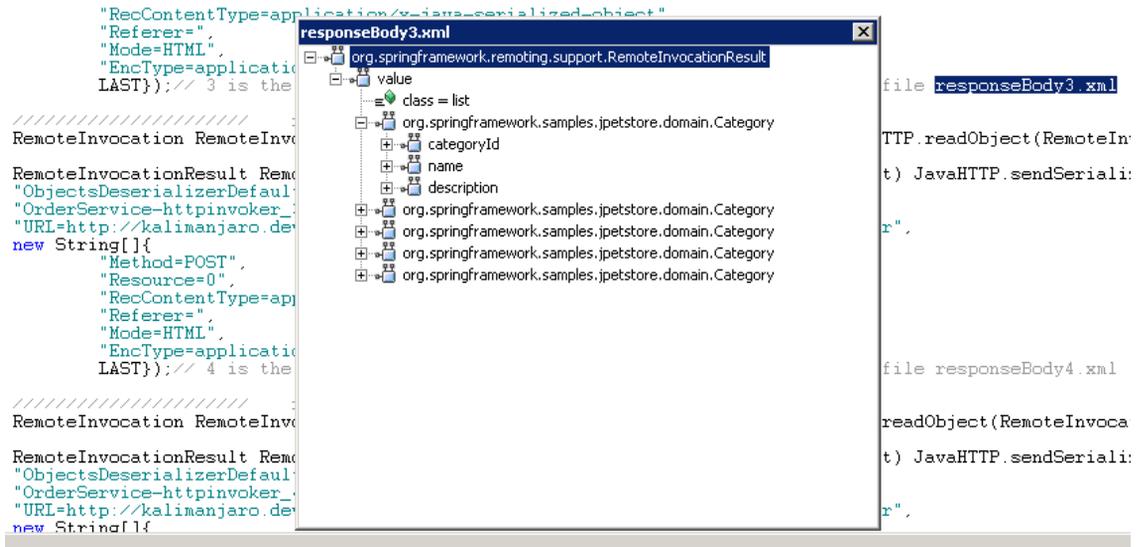
The Java over HTTP protocol is designed to record java-based applications and applets. It produces a Java language script using web functions. This protocol is distinguished from other Java protocols in that it can record and replay Java remote calls over HTTP.

Viewing Responses and Requests in XML Format

For each request and response, you can view the corresponding XML that represents the binary java object during the recording phase.

To view xml data:

- 1 Locate the target request or response section in the code. Right-click the commented **RequestBodyX.xml** or **ResponseBodyX.xml**.
- 2 Select **View XML**. The XML is displayed in a separate window.



The screenshot shows a code editor on the left with XML data. A separate window titled "responseBody3.xml" is open in the center, displaying the XML content. The XML content is as follows:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" type="text/xml">
<org.springframework.remoting.support.RemoteInvocationResult>
  <value>
    <class>list</class>
    <org.springframework.samples.jpeteststore.domain.Category>
      <categoryId>
      <name>
      <description>
    </org.springframework.samples.jpeteststore.domain.Category>
    <org.springframework.samples.jpeteststore.domain.Category>
    <org.springframework.samples.jpeteststore.domain.Category>
    <org.springframework.samples.jpeteststore.domain.Category>
  </value>
</org.springframework.remoting.support.RemoteInvocationResult>

```

The code editor on the left shows the following XML data:

```

"RecContentType=application/x-java-serialized-object"
"Referer="
"Mode=HTML"
"EncType=application/x-www-form-urlencoded"
LAST}); // 3 is the
RemoteInvocation RemoteInvoc
RemoteInvocationResult Remo
"ObjectsDeserializerDefault
"OrderService-httpinvoker_
"URL=http://kalimanjaro.de
new String[]{
  "Method=POST",
  "Resource=0",
  "RecContentType=app
  "Referer="
  "Mode=HTML"
  "EncType=applicati
  LAST}); // 4 is the
RemoteInvocation RemoteInvoc
RemoteInvocationResult Remo
"ObjectsDeserializerDefault
"OrderService-httpinvoker_
"URL=http://kalimanjaro.de
new String[]f

```

Tasks

How to Record with Java over HTTP

To recording with Java over HTTP you must specify which .jar files to use in order to deserialize the recorded data.

This task describes how to locate the relevant .jar files and add them to the classpath.

This task includes the following steps:

- ▶ "Recording Java Applets" on page 722
- ▶ "Recording Local Java Applications" on page 723

Recording Java Applets

If your application uses Java Applets, you need to find the relevant .jar files and enable them in the classpath.

To locate the relevant .jar files:

- 1** Clear the JAR cache by selecting **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Delete Files**.
- 2** Open your application and perform a few business processes to repopulate the JAR cache with .jar files from your application. When you are finished, close your application
- 3** Select **Control Panel > Java > General Tab > Temporary Internet Files > View**. This lists the JAR cache and should only contain the .jar files used by your application.
- 4** Download the files. Try the options in the order in which they appear. When you succeed, proceed to the next step to Add the .jar files to the classpath.
 - a** Option 1: For each .jar file, go to the listed URL and download the file. If you cannot download one or all of the .jar files, continue with the next option.

- b** Option 2: Clear the cache again by selecting **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Delete Files**. Open your application again and perform a few business processes. Do not close your application. Open the Java Console . There should be a message for each .jar file telling you the location it is stored in a temporary file on your computer. The files are usually hashed and don't have .jar extensions. Change the name (including changing each extension to .jar) and copy the file to a known location.
 - c** Option 3: If the files don't show up in the Java console, locate the temporary directory as listed in **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Location**. Open the specified location and rename all the files in the sub-folders to .jar. Do not rename all the files in the main folder.
- 5** Add the .jar files to the classpath in the **Recording Options > Java Environment Settings > Classpath** node. For more information, see "Java Classpath Node" on page 373.

Recording Local Java Applications

If you are recording a local Java application (not an applet), all of the .jar files already exist on your computer.

To locate the relevant .jar files:

- 1** Look in the batch file that launched the application. All of the .jar files that are referenced should be added to the classpath.
- 2** If you cannot locate or understand the batch file, add all of the .jar files from the application folder and subfolders to the classpath.
- 3** Add the .jar files to the classpath in the **Recording Options > Java Environment Settings > Classpath** node. For more information, see "Java Classpath Node" on page 373.

How to Debug Java over HTTP scripts

This task describes how to debug Java over HTTP scripts by comparing the request and response data from both the record and replay stages.

This task includes the following steps:

- "Add arguments to the VM Param Node" on page 724
- "Compare record and replay data" on page 724
- "Remove arguments before load testing" on page 725

1 Add arguments to the VM Param Node

Select **Vuser** > **Run-Time Settings** > **Java VM** node. In the **Additional VM Parameters** field, enter the following string:

```
-DdumpServerRequests=true -DdumpServerResponses=true
```

2 Compare record and replay data

While in script view, right-click and select **Open Script Directory**. The data from the recording phase is in the main directory. The data from the replay phase is in the replay directory.

The files that follow the format `RequestBodyX` contain the request data. The files that follow the format `ResponseBodyX` contain the response data.

To compare the record and replay data for the purposes of debugging, compare the files with identical names from the recording and replay phases. For example, compare the `RequestBody1` file from the main directory (recording phase) to the `RequestBody1` file from the replay directory. Normally, the files should be identical. Cases where the files are not identical may indicate problems in the script.

3 Remove arguments before load testing

Return to the Java VM node and the items you added to the Additional VM Parameters field.

How to Insert Parameters into Java over HTTP Scripts

Parameter functions can be added for each response or request body text in a specific location. This location is indicated by a blank line usually one to two lines below the start of the response or request body. In the example below, parameter functions can be added to the blank lines in each requestBody section.

```

//////////////////////////////////// requestBody2.xml //////////////////////////////////////
RemoteInvocation RemoteInvocation_getUsernameList2 =
    (RemoteInvocation) JavaHTTP.readObject(RemoteInvocationBA0);
//INSERT PARAMETERIZATION AND CORRELATION CODE HERE
RemoteInvocationResult RemoteInvocationResult_ArrayList2 =
    (RemoteInvocationResult) JavaHTTP.sendSerialized(RemoteInvocation_getUsernameList
"ObjectsDeserializerDefaultImpl",
"OrderService-httpinvoker",
"URL=http://kalimanjaro.devlab.ad:8080/jpetstore/remoting/OrderService-httpinvoker",
new String[]{
    "Method=POST",
    "Resource=0",
    "RecContentType=application/x-java-serialized-object",
    "Referer=",
    "Mode=HTML",
    "EncType=application/x-java-serialized-object",
    LAST}); // 2 is the number of the header file, record time response is at file

//////////////////////////////////// requestBody3.xml //////////////////////////////////////
RemoteInvocation RemoteInvocation_getCategoryList3 =
    (RemoteInvocation) JavaHTTP.readObject(RemoteInvocationBA1);
//INSERT PARAMETERIZATION AND CORRELATION CODE HERE
RemoteInvocationResult RemoteInvocationResult_ArrayList3 =
    [(RemoteInvocationResult) JavaHTTP.sendSerialized(RemoteInvocation_getCategoryList
"ObjectsDeserializerDefaultImpl",
"OrderService-httpinvoker_2",
"URL=http://kalimanjaro.devlab.ad:8080/jpetstore/remoting/OrderService-httpinvoker",
new String[]{
    "Method=POST",
    "Resource=0",
    "RecContentType=application/x-java-serialized-object",
    "Referer=",

```

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Java over HTTP protocol.

Limitations

- ▶ JDK 1.5 or higher is required.
- ▶ Lazy evaluating objects are not supported, for example hibernate in lazy mode.
- ▶ If there are stateful serialization mechanisms on the application server, this can interfere with LoadRunner deserialization and result in unserialized data and unexpected errors.
- ▶ The following menu items are not available for this protocol:
 - ▶ Insert > New Step / Start Transaction / End Transaction / Rendezvous

Disable Exception Error Checking

If you are receiving exception errors and you are sure that the error is irrelevant, VuGen allows you to disable all such error messages. To do this, select **Vuser > Run-Time Settings > Java VM** node. In the **Additional VM Parameters** field, and append the following string to the end of the current entry:

```
-DvalidateServerResponse=false
```

Additionally, you can change the error checking behavior of a specific step by adding a closing argument to the `sendSerialized` function in script view. For more information, see the *HP LoadRunner Online Function Reference*.

Cannot Correlate Private Object Members

When you need to correlate or parameterize data that is a private member of an object, you can use the `Irapi.Ir2.fieldSetter` and `Irapi.Ir2.fieldGetter` functions.

```

RemoteInvocation RemoteInvocation2 = (RemoteInvocation)
JavaHTTP.readObject(RemoteInvocationBA0);
RemoteInvocation.methodName="applyToSchool";
Student student=RemoteInvocation.arguments[0];

Map grades=Ir2.fieldGetter(student,"grades");//grades is a private member of Student
grades.put("Math", "95");
Ir2.fieldSetter(student,"super.name","Tom");
//Student class inherits the name field from Person. name field is a string
Ir2.fieldSetter(student,"super.ID", "98764321");
//Student class inherits the ID field from Person. ID field is an int

RemoteInvocationResult RemoteInvocationResult_ArrayList2 =
(RemoteInvocationResult) JavaHTTP.sendSerialized(RemoteInvocation2, 2,
"ObjectsDeserializerDefaultImpl",....

```


24

LDAP Protocol

This chapter includes:

Concepts

- ▶ LDAP Protocol Overview on page 730
- ▶ LDAP Protocol Example Script on page 730
- ▶ Defining Distinguished Name Entries on page 732
- ▶ LDAP Connection Options on page 733

Concepts

LDAP Protocol Overview

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 732.

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.

LDAP Protocol Example Script

All LDAP functions come in pairs—one for global sessions and one where you can indicate a specific session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the **ex** suffix. For example, **mldap_logon** logs on to the LDAP server globally, while **mldap_logon_ex** logs on to the LDAP server for a specific session.

In the following example, the user logs on to an LDAP server, ldap1. It adds an entry and then renames the OU attribute from Sales to Marketing.

```
Action()
{
    // Logon to the LDAP server
    mldap_logon("Login",
               "URL=ldap://johnsmith:tiger@ldap1:80",
               LAST);

    // Add an entry for Sally R. Jones
    mldap_add("LDAP Add",
             "DN=cn=Sally R. Jones,OU=Sales, DC=com",
             "Name=givenName", "Value=Sally", ENDITEM,
             "Name=initials", "Value=R", ENDITEM,
             "Name=sn", "Value=Jones", ENDITEM,
             "Name=objectClass", "Value=contact", ENDITEM,
             LAST);

    // Rename Sally's OU to Marketing
    mldap_rename("LDAP Rename",
                "DN=CN=Sally R. Jones,OU=Sales,DC=com",
                "NewDN=OU=Marketing",
                LAST);

    // Logout from the LDAP server
    mldap_logoff();

    return 0;
}
```

Defining Distinguished Name Entries

The LDAP API references objects by its **distinguished name** (DN). A DN is a sequence of **relative distinguished names** (RDN) separated by commas.

An RDN is an attribute with an associated value in the form attribute=value. The attribute names are not case-sensitive. The following table lists the most common RDN attribute types.

String	Attribute Type
DC	domainComponent
CN	commonName
OU	organizationalUnitName
O	organizationName
STREET	streetAddress
L	localityName
ST	stateOrProvinceName
C	countryName
UID	userid

The following are examples of distinguished names:

```
DN=CN=John Smith,OU=Accounting,DC=Fabrikam,DC=COM
DN=CN=Tracy White,CN=admin,DC=corp,DC=Fabrikam,DC=COM
```

The following table lists reserved characters that cannot be used in an attribute value.

Character	Description
	space or # character at the beginning of a string
	space character at the end of a string

,	comma
+	plus sign
"	double quote
\	backslash
<	left angle bracket
>	right angle bracket
;	semicolon

To use a reserved character as part of an attribute value, you must precede it with an escape character, a backslash (\). If an attribute value contains other reserved characters, such as the equal sign (=) or non-UTF-8 characters, you must encode it in hexadecimal format—a backslash followed by two hex digits.

The following are examples of DNs that include escaped characters. The first example is an organizational unit name with an embedded comma; the second example is a value containing a carriage return.

```
DN=CN=Bitwise,OU=Docs\, Support,DC=Fabrikam,DC=COM
DN=CN=Before\0DAfter,OU=Test,DC=North America,DC=Fabrikam,DC=COM
```

LDAP Connection Options

Using the `ldap_logon[_ex]` function, you control the way you login to the LDAP server.

When specifying the URL of the LDAP server, you specify how to connect and with what credentials.

When specifying the server's URL, use the following format:

```
ldap[s][username:[password]@][server[:port]]
```

The following table shows several examples of connections to LDAP servers.

Syntax	Description
ldap://a:b@server.com:389	Connects to the server (to 389 port) and then binds with username "a" , password "b"
ldap://:@server.com	Connects to server (to default unsecured port 389) then binds anonymously with a NULL username and password
ldaps://a:@server.com	Connects to server (to default secured port 636)and then binds with username "a", password ""
ldap://@server.com, ldap://server.com	Connects to server without binding
ldap://a:b@	Binds with username "a", password "b", executing a bind on the existing session without reconnecting
ldap://:@	Binds anonymously with a NULL username and password (executes bind on existing session without reconnecting)

You can also specify LDAP modes or SSL certificates using the following optional arguments:

- ▶ **Mode.** The LDAP call mode: *Sync* or *Async*
- ▶ **Timeout.** The maximum time in seconds to search for the LDAP server.
- ▶ **Version.** The version of the LDAP protocol version 1,2, or 3
- ▶ **SSLCertDir.** The path to the SSL certificates database file (cert8.db)
- ▶ **SSLKeysDir.** The path to the SSL keys database file (key3.db)
- ▶ **SSLKeyNickname.** The SSL key nickname in the keys database file
- ▶ **SSLKeyCertNickname.** The SSL key's certificate nickname in the certificates database file
- ▶ **SSLSecModule.** The path to the SSL security module file (secmod.db)
- ▶ **StartTLS.** Requires that the StartTLS extension's specific command must be issued in order to switch the connection to TLS (SSL) mode

For detailed information about these arguments, see the *Online Function Reference* (**Help > Function Reference**).

25

Mailing Service Protocols

This chapter includes:

Concepts

- ▶ Mailing Service Protocols Overview on page 738
- ▶ IMAP Protocol Overview on page 738
- ▶ MAPI Protocol Overview on page 739
- ▶ POP3 Protocol Overview on page 741
- ▶ SMTP Protocol Overview on page 742

Concepts

Mailing Service Protocols Overview

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.

IMAP Protocol Overview

IMAP Vuser script functions record the Internet Mail Application Protocol. Recording is not supported for this 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;
}

```

MAPI Protocol Overview

MAPI Vuser script functions record activity to and from an MS Exchange server. Each MAPI function begins with a **mapi** prefix. For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

Note: To run MAPI scripts, you must define a mail profile on the machine running the script. For example, install Outlook Express, set it as the default mail client, and create a mail account. Alternatively, install Microsoft Outlook, set it as the default mail client, create a mail account and create a mail profile. To create a mail profile in Microsoft Outlook, select **Settings > Control Panel > Mail > Show Profiles** and add a mail profile.

In the following example, the **mapi_send_mail** function sends a sticky note through an MS Exchange server.

```
Actions()
{
    mapi_logon("Logon",
              "ProfileName=John Smith",
              "ProfilePass=Tiger",
              LAST);
    //Send a Sticky Note message
    mapi_send_mail("SendMail",
                  "To=user1@techno.merc-int.com",
                  "Cc=user0002t@techno.merc-int.com",
                  "Subject=<GROUP>:<VUID> @ <DATE>",
                  "Type=Ipm.StickyNote",
                  "Body=Please update your profile today.",
                  LAST);

    mapi_logout();
    return 1;
}
```

POP3 Protocol Overview

POP3 Vuser script functions emulate actions using the Post Office Protocol, POP3. Each function begins with a **pop3** prefix. For detailed syntax information on these functions, see the *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;
}

```

SMTP Protocol Overview

SMTP Vuser script functions emulate the Single Mail Transfer Protocol traffic. Each SMTP function begins with an **smtp** prefix. For detailed syntax information on these functions, see the *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=Smtptest User 0001",
        NULL);

    smtp_send_mail("SendMail",
        "To=user0002t@merc-int.com",
        "Subject=MIC Smtptest: 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;
}

```

26

Microsoft .NET Protocol

This chapter includes:

Concepts

- ▶ Microsoft .NET Protocol Overview on page 744
- ▶ Viewing Data Sets and Grids on page 744
- ▶ Recording WCF Duplex Communication on page 746
- ▶ Recording Dual HTTP Bindings on page 752
- ▶ Asynchronous Calls on page 752
- ▶ Connection Pooling on page 755
- ▶ Debugging Microsoft .NET Scripts on page 756
- ▶ Microsoft .NET Filters Overview on page 757
- ▶ Microsoft .NET Filters - Advanced on page 759
- ▶ Guidelines for Setting Microsoft .NET Filters on page 760

Tasks

- ▶ How to Configure Application Security and Permissions on page 765

Reference

Troubleshooting and Limitations on page 767

Concepts

Microsoft .NET Protocol Overview

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>.

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**.

	FLIGHT NUMBER	DEPARTURE INITIALS	DEPARTURE	DAY OF WEEK	ARRIVAL INITIALS	ARRIVAL	DEPARTURE T
1	5709	DEN	Denver	Saturday	LAX	Los Angeles	05:21 PM
2	3636	DEN	Denver	Saturday	LAX	Los Angeles	01:45 PM
3							
4							
5							
6							
7							
8							
9							

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.

Recording WCF Duplex Communication

WCF (Windows Communication Foundation) is a programming model that unifies Web Services, .NET Remoting, Distributed Transactions, and Message Queues into a single Service-oriented programming model for distributed computing.

WCF creates a proxy object to provide data for the service. It also marshalls the data returned by the service into the form expected by the caller.

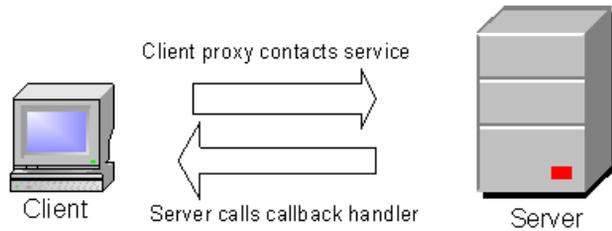
In addition to general support for the WCF environment, VuGen provides specialized support for applications that use WCF's duplex communication. In duplex communication, the client proxy contacts the service, and the service invokes the callback handler on the client machine. The callback handler implements a callback interface defined by the server. The server does not have to respond in a synchronous manner—it independently determines when to respond and invoke the callback handler.

Communication Between Client and Server

The communication between the client and server is as follows:

- The server defines the service contract and an interface for the callback.
- The client implements the callback interface defined by the server.

- The server calls the callback handler in the client whenever needed.



When trying to record and replay duplex communication, you may encounter problems when the script calls the original callback methods. By default, the callback handlers are not included in the filter. You could customize the filter to include those callback handlers. However, the standard playback would be ineffective for a load test, since many of the callbacks are local operations such as GUI updates. For an effective load test you cannot replay the original callback method invoked by the server.

VuGen's solution is based on replacing the original callback handler with a dummy implementation. This implementation performs a typical set of actions that you can customize further for your application.

You instruct VuGen to replace the original callbacks by activating the **Generate Dummy Callback Handler** recording option. For more information, see "Remote Objects Property" on page 386

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:

```

namespace script
{
    using System;
    using System.Threading;
    using System.ServiceModel;
    using System.Collections.Generic;
    using Mercury.LoadRunner.DotNetProtocol.Repl;

    ///-----
    /// Helper class for handling duplex callbacks
    /// This class is the base class for the dummy
    /// used when "Generate dummy callback handler"
    ///-----
    public class VuserDuplexCallbackHelper<ID, RE>
    {
        // Initialize LoadRunner API
        protected LoadRunner.LrApiTr = new LoadR

        // Synchronization event for responses
        private AutoResetEvent waitForResponseEvt
    }

```

The Helper file serves as a general template for working with duplex callback handlers. It serves as a base class for the implementation of the callback.

The second file, **Callback_Name**, contains the implementation of the callback. The name of the callback implementation class is **Vuser<xxxx>** where *xxxx* is the name of the callback interface and it inherits from the **VuserDuplexCallbackHelper** class defined in the Helper file. VuGen creates separate implementation files for each interface.

This file performs two primary tasks:

- **Set Response.** It stores the data that came from the server in a map. It stores them with sequential IDs facilitating their retrieval. This method is called from the implementation of the callback interface. The following sample code demonstrates the dummy implementation of a callback method named **Result**. The method's arguments are stored in the map as an object array.

```
// -----
public virtual void Result(string operation, double result) {
    // Add here your own callback implementation and set the response data
    SetResponse(responseIndex++, new object[] {
        operation,
        result});
}
```

- **Get Response.** Waits for the next response to arrive. The implementation of `GetNextResponse` retrieves the next response stored in the map using a sequential indexer, or waits until the next response arrives.

The script calls `GetNextResponse` at the point that the original callback handler was called during recording. At that point, the script prints a warning:

```
// Wait here for the next response.
// The original callback during record was:
```

Replacement of the Callback in the Script

When you enable the **Dummy Callback** option (enabled by default), VuGen replaces the original duplex callback handlers with dummy implementations. The dummy implementation is called `Vuser <Callback Name>`. At the point of the original callback handler, the script prints a warning indicating that it was replaced.

Customizing the Dummy Implementation

You can modify the implementation file to reflect your environment. This section contains several suggested customizations.

Timeouts

The default timeout for which the callback waits for the next response is 60000 msec, or one minute. To use a specific timeout, replace the call to **GetNextResponse** with the overloading method which gets the timeout as an argument as shown below. This method is implemented in the callback implementation file *<Callback_Name>* listed in the left pane after the **DuplexCallbackHelper** file.

```
// Get the next response.
// This method waits until receiving the response from the server
// or when the specified timeout is exceeded.
public virtual object GetNextResponse(int millisecondsTimeout) {
    return base.GetResponse(requestIndex++, millisecondsTimeout);
}
```

To change the default threshold for all callbacks, modify the **DuplexCallbackHelper** file.

```
// Default timeout threshold while waiting for response
protected int millisecondsTimeoutThreshold = 60000;
```

Key Identifier

Many applications assign key identifiers to the data, which connects the request and response to one another. This allows you to retrieve the data from the map using its ID instead of the built-in incremental index. To use a key identifier instead of the index, modify the file *<Callback_Name>* replacing the first base template parameter, **named ID**, with the type of your key identifier. For example, if your key identifier is a string you may change the first template argument from **int** to **string**:

```
public class VuserXXX : VuserDuplexCallbackHelper<string, object>
```

In addition, you may remove the implementation of `GetNextResponse()` and replace it with calls to `GetResponse(ID)` defined in the base class.

Return Values

By default, since VuGen supports *OneWay* communication, the implementation callback does not return any value or update an output parameter when it is invoked.

```
public virtual void Result(string operation, double result) {
    // Add here your own callback implementation and set the response data
}
```

If your application requires that the callback return a value, insert your implementation at that point.

Get Response Order

In VuGen's implementation, a blocking method waits for each response. This reflects the order of events as they occurred during recording—the server responded with data. You can modify this behavior to execute without waiting for a response or to implement the blocking only after the completion of the business process.

Find Port

The **FindPort** method in the Helper file is a useful utility that can be used in a variety of implementations. The Helper class uses this method to find unique ports for running multiple instances of the script. You can utilize this utility method for other custom implementations.

Recording Server Hosted By Client Applications

If the communication in your system is a server hosted by a client, VuGen's default solution for duplex communication will not be effective. In server hosted by client environments, it is not true duplex communication since the client opens the service and does not communicate through the Framework. For example, in queuing, the client sends a message to the service and opens a response queue to gather the responses.

To emulate a server hosted by a client, use the pattern depicted in the above solution—replace the original response queues with dummy callbacks and perform synchronization as required. For more information, contact HP support.

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:

For more information, see "Microsoft .NET Recording Node" on page 382

```
#warning: Code Generation Warning
// Override the original client base address with a unique port number
DualProxyHelper.SetUniqueClientBaseAddress<XXXX>(YYYYYY);
```

Asynchronous Calls

When VuGen records asynchronous calls on remote objects, you can specify how the calls are handled in the "Microsoft .NET Recording Node" on page 382. These options are particularly relevant for .NET Remoting and WCF environments.

You can configure VuGen to one of the following options:

- **Call original callbacks by default.** Uses the recorded application's original callback when generating and replaying the script. If the callback method is explicitly excluded by a filter, the callback will be excluded even if you enable this option. If your callbacks perform actions that are not directly related to the business process, such as updating the GUI, then make sure to disable this option.
- **Generate asynchronous callbacks.** This option defines how VuGen will handle callbacks when the original callbacks are not recorded. This is relevant when the above option, **Call original callbacks** is disabled or when the callbacks are explicitly excluded.

When you enable this option, it creates a dummy method which will be called during replay instead of the original callback. This dummy callback will be generated in the **callbacks.cs** section of the script.

When you disable this option, VuGen inserts a NULL value for the callback and records the events as they occur.

The following segment shows script generation for a Calculator client, when **Generate asynchronous callbacks** is enabled.

```
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 "Microsoft .NET Recording Node" on page 382

Connection Pooling

ADO.NET providers deploy a feature called **connection pooling** which can significantly influence load test accuracy. Whenever only one app domain is used for all Vusers, connection pooling is turned on—.NET Framework keeps the database connections open and tries to reuse them when a new connection is requested. Since many Vusers are executed in the context of a single application domain, they may interfere with one another. Their behavior will not be linear and that may decrease their accuracy.

In the .NET run-time settings, the AppDomain Per Vuser property enables execution of each Vuser in a separate app domain (true by default). This means that there is connection pooling in the scope of each Vuser, but the Vusers will not interfere with one another. This setting provides more accuracy, but lower scalability.

If you disable this option, you need to manually disable connection pooling for the database.

The following table describes how to manually disable connection pooling:

Provider	Option
.NET Framework Data Provider for SQL Server	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for Oracle	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for ODBC	Connection pooling is managed by an ODBC Driver Manager. To enable or disable connection pooling, use the ODBC Data Source Administrator (found in Control Panel or the Administrative Tools folder). The Connection Pooling tab allows you to specify connection pooling parameters for each of the installed ODBC drivers.
.NET Framework Data Provider for OLE DB	"OLE DB Services=-2"
Oracle Data Provider for .NET	"pooling=false"
Adaptive Server Enterprise ADO.NET Data Provider	"Pooling=False"

Debugging Microsoft .NET Scripts

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.

Microsoft .NET Filters Overview

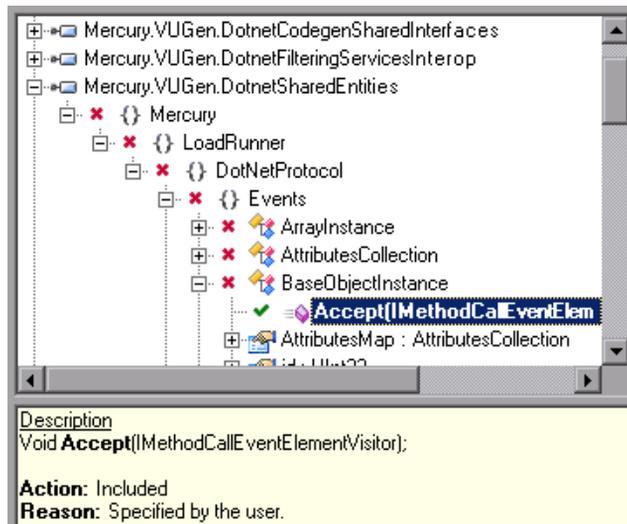
Recording filters indicate which assemblies, interfaces, namespaces, classes, or methods to include or exclude during the recording and script generation.

By default, VuGen provides built-in system filters for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation). These filters were designed to include the relevant interfaces for standard ADO.NET, Remoting, Enterprise Services, and WCF. VuGen also allows you to design custom filters.

Custom filters provide several benefits:

- ▶ **Remoting.** When working with .NET Remoting, it is important to include certain classes that allow you to record the arguments passed to the remote method.
- ▶ **Missing Objects.** If your recorded script did not record a specific object within your application, you can use a filter to include the missing interface, class or method.
- ▶ **Debugging.** If you receive an error, but you are unsure of it's origin, you can use filters to exclude methods, classes, or interfaces in order to pin-point the problematic operation.
- ▶ **Maintainability.** You can record script in higher level, make script more easy to maintain and to correlate.

A filter manager lets you manipulate existing custom filters. It displays the assemblies, namespaces, classes, methods, and properties in a color-coded tree hierarchy.



The bottom pane provides a description of the assembly, namespace, class, method, property, or event. It also indicates whether or not it is included or excluded and a reason for the inclusion or exclusion.

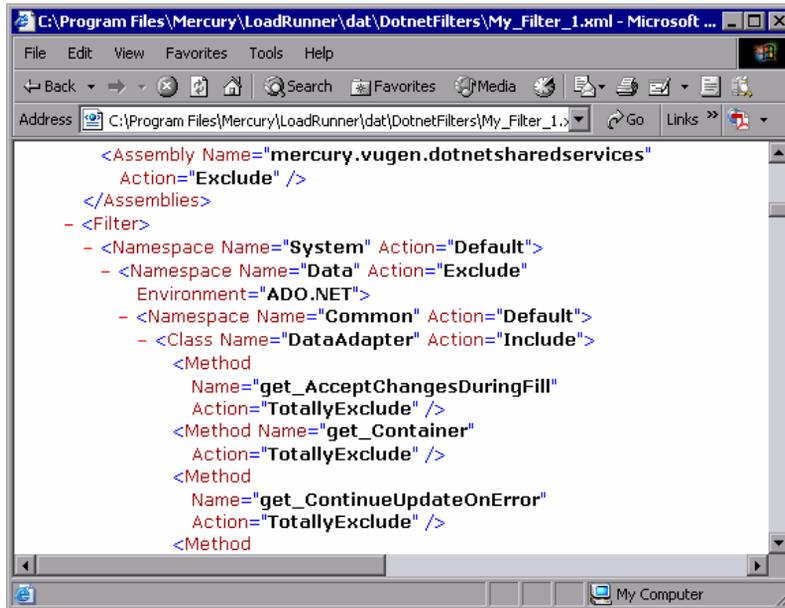
Microsoft .NET Filters - Advanced

In the Filter Manager's tree hierarchy, it only displays public classes and methods. It does not show non-public classes or delegates.

You can add classes or methods that are not public by manually entering them in the filter's definition file.

The filter definition files, **<filter_name>.xml** reside in the `dat\DotnetFilters` folder of your installation. The available Action properties for each element are: **Include**, **Exclude**, or **Totally Exclude**. For more information, see "Filter Manager" on page 377.

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.



```

<Assembly Name="mercury.vugen.dotnetshareservices"
  Action="Exclude" />
</Assemblies>
- <Filter>
  - <Namespace Name="System" Action="Default">
  - <Namespace Name="Data" Action="Exclude"
    Environment="ADO.NET">
  - <Namespace Name="Common" Action="Default">
  - <Class Name="DataAdapter" Action="Include">
    <Method
      Name="get_AcceptChangesDuringFill"
      Action="TotallyExclude" />
    <Method Name="get_Container"
      Action="TotallyExclude" />
    <Method
      Name="get_ContinueUpdateOnError"
      Action="TotallyExclude" />
    <Method
  
```

For example, suppose Function A calls function B. If Function A is **Excluded**, then when the service calls Function A, the script will include a call to Function B. However, if function A is **Totally Excluded**, the script will not include a call to Function B. Function B would only be recorded if called directly—not through Function A.

VuGen saves a backup copy of the filter as it was configured during the recording, **RecordingFilterFile.xml**, in the script's **data** folder. This is useful if you made changes to the filter since your last recording and you need to reconstruct the environment.

Guidelines for Setting Microsoft .NET 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 "Filter Manager" on page 377.

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 761.
- 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, see "How to Correlate Scripts - Microsoft .NET" on page 212.

Tasks

How to Configure Application Security and Permissions

A Security Exception that occurs while recording an application is usually due to a lack of permissions—the recording machine does not have sufficient permissions to record the application. This is common where your application is not local, but on the Intranet or network.

To solve this problem, you need to allow the recording machine to access the application and the script with Full Trust.

One solution is to copy the application and save your script locally, since by default, users have Full Trust permissions to all local applications and folders.

An additional solution is to create new code groups that gives Full Trust to each application folder, and the script folder.

To grant Full Trust permissions to a specific folder (Visual Studio NOT installed):

- 1** From the command prompt, run the caspol.exe application.
- 2** Set the desired permission.

To grant Full Trust permissions to a specific folder (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).

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Microsoft .NET protocol.

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.

27

Oracle NCA Protocol

This chapter includes:

Concepts

- ▶ Oracle NCA Protocol Overview on page 770
- ▶ Oracle NCA Protocol Example Scripts on page 771
- ▶ Oracle NCA Record and Replay Tips on page 772
- ▶ Pragma Mode on page 773

Tasks

- ▶ How to Enable the Recording of Objects by Name on page 776
- ▶ How to Launch Oracle Applications via the Personal Home Page on page 779

Reference

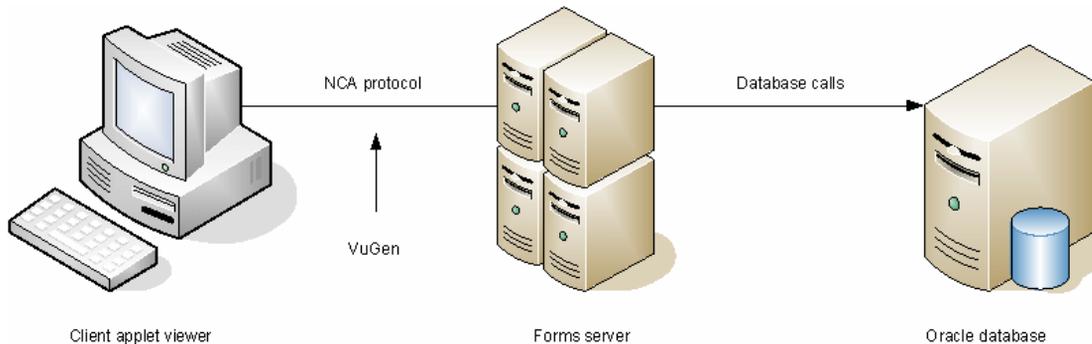
Troubleshooting and Limitations on page 781

Concepts

Oracle NCA Protocol Overview

Oracle NCA is a protocol that handles communication with the Oracle Forms server. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer. This eliminates the need for client software and allows you to perform database actions from all platforms that support the applet viewer.

The NCA environment is a three-tier environment. The user first sends an http call from his browser to a Web server. This call accesses the startup HTML page which invokes the Oracle Applications applet. The client (applet viewer) communicates through the proprietary NCA protocol with the application server (Oracle Forms server) which then submits information to the database server.



VuGen records and replays the NCA communication between the client and the Forms server (application server).

The Oracle NCA protocol is commonly used as a multi protocol in combination with Web (HTTP/HTML) or Web (Click and Script). This is the recommended way to record with Oracle NCA. If you are using Oracle NCA as a single protocol, web events are recorded but steps are not generated (or replayed) by default.

If you initially created a single protocol script for Oracle NCA, and at a later stage you require the Web functions for testing, you can regenerate your script in VuGen to add the Web functions, without having to re-record the session. You indicate this from the Protocols node in the Recording Options.

Oracle NCA Protocol Example Scripts

In the following example, the user selected an item from a list (`nca_list_activate_item`), pressed a button (`nca_button_press`), retrieved a list value (`nca_lov_retrieve_items`), and performed a click in an edit field (`nca_edit_click`). The logical names of the objects are the parameters of these functions.

```
...
nca_lov_select_item("Responsibilities","General Ledger, Vision Operations");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","+ Journals");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","  Enter");
nca_button_press("GLXJEENT.TOOLBAR.LIST.0");
nca_lov_find_value("Batches","");
nca_lov_retrieve_items("Batches",1,9);
nca_lov_select_item("Batches","AR 1020 Receivables 2537: A 1020");
nca_edit_click("GLXJEENT.FOLDER_QF.BATCH_NAME.0");
...
```

In certain tests, such as those performed on Oracle Configurator applications, information returned by one function is required throughout the session. VuGen automatically saves the dynamic information to a parameter, by inserting a `web_reg_save_param` function into the script. In the following example, the connection information is saved to a parameter called `NCAJServSessionID`.

```
web_reg_save_param ("NCAJServSessionId", "LB=\r\n\r\n", "RB=\r",
  LAST);
web_url("f60servlet",
  "URL=http://usscifforms05.sfb.na/servlet/f60servlet?config
  =mult", LAST);
```

In the above example, the right boundary is `\r`. The actual right boundary may differ between systems.

Note: We recommend that the user not modify the **web_reg_save_param** parameters if it was generated automatically. Alternatively, you can manually add a new **web_reg_save_param** function or add a new correlation rule.

Oracle NCA Record and Replay Tips

When recording an Oracle NCA Vuser script, follow these guidelines:

- ▶ We recommend installing Jinitiator before recording a script.
- ▶ Close all browsers before you begin recording.
- ▶ Due to a Netscape limitation, you cannot launch an Oracle NCA session within Netscape when another Netscape browser is already running on the machine.
- ▶ 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 "Script Sections" on page 34.
- ▶ VuGen supports the recording of Oracle Forms applications using the Forms Listener Servlet in multi-protocol mode. The application server uses the **Forms Listener Servlet** to create a runtime process for each client. The runtime process, **Forms Server Runtime**, maintains a persistent connection with the client and sends information to and from the server.

To support Forms 4.5 in replay, modify the **mdrv_oracle_nca.dat** file in the **dat > mdrv** directory to match the following example:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp110.dll
WIN95_EXT_LIBS=ncarp110.dll
LINUX_EXT_LIBS=liboranca.so
SOLARIS_EXT_LIBS=liboranca.so
HPUX_EXT_LIBS=liboranca.sl
AIX_EXT_LIBS=liboranca.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api
```

To restore Forms support for versions later than 4.5, restore the original values.

Pragma Mode

The client side of the Oracle NCA Vuser can be configured to send an additional header to the server named **Pragma**. The header is a counter that behaves in the following way: the initial message of the NCA handshake has a value of 1.

The messages that follow the handshake are counted, beginning with 3. The counter is incremented by 1 for each message sent by the client.

If the message received from the server is the type `plain\text` and the body of the message begins with `ifError:###00`, the client sends a 0 byte message to the server and the Pragma value changes its sign to a minus. This sign changes back after the client succeeds in receiving the information from the server.

Recording of the Pragma header is only supported in the multi-protocol mode (Oracle NCA and Web). You can identify the Pragma mode within the script's default.cfg file. When operating in Pragma mode, the UseServletMode is set to 2.

```
[HttpConnectMode]
UseHttpConnectMode=1
RelativeURL=<NCAJServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see "Oracle NCA Client Emulation Node" on page 480.

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"
...
```

Tasks

How to Enable the Recording of Objects by Name

When recording an Oracle NCA script, you must record the session using object names instead of the standard object ID. If the script is recorded using the object ID, replay may fail because the ID is generated dynamically by the server and may differ between iterations. You can verify that your script is being recorded with object names by examining the `nca_connect_server` function.

```
nca_connect_server("199.35.107.119","9002"/*version=11i*/,"module=/d1/oracle/
visappl/fnd/11.5.0/forms/US/FNDSCSGN userid=APPLSYSPUB/PUB@VIS
fndnam=apps record=names ");
```

If the `record=names` argument does not appear in the `nca_connect_server` function, you may be recording object IDs. You can instruct VuGen to record object names by modifying one of the following:

- "Startup HTML File" on page 776
- "Forms Configuration File" on page 777
- "URL to Record" on page 778

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.

To enable the recording of object names using the startup HTML file:

- 1 Edit the startup file that is called when the applet viewer begins by modifying the line shown below.

```
<PARAM name="serverArgs ... fndnam=APPS">
```

- 2 Add the Oracle key **record=names** as shown below.

```
<PARAM name="serverArgs ... fndnam=APPS record=names">
```

Forms Configuration File

If the application has a startup HTML file that references a Forms Web CGI configuration file **formsweb.cfg** (a common reference), you may encounter problems if you add **record=names** to the Startup file. In this situation, you should modify the configuration file.

To enable the recording of object names using the configuration file:

- 1 Go to the Forms Web CGI configuration file.
- 2 Define a new parameter in this file (see sample Web CGI configuration file below for this change).

```
serverApp=forecast
serverPort=9001
serverHost=easgdev1.dats.ml.com
connectMode=socket
archive=f60web.jar
archive_ie=f60all.cab
xrecord=names
```

- 3 Open the startup HTML file and locate **PARAM NAME="serverArgs"**.
- 4 Add the variable name as an argument to the ServerArgs parameter, for example, **record=%xrecord%** as shown below.

```
<PARAM NAME="serverArgs" VALUE="module=%form% userid=%userid%
%otherParams% record=%xrecord%">
```

- 5 Alternatively, you can edit the **basejini.htm** file in the Oracle Forms installation 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 **basejini.htm** file add the following line to the parameter definitions:

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the <EMBED> tag, add the following line:

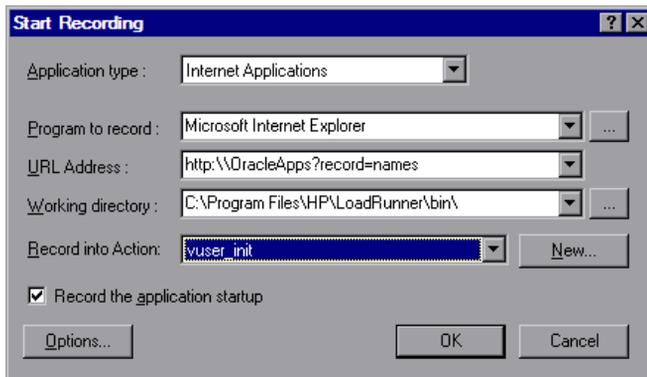
```
serverApp="%serverApp%"
logo="%logo%"
imageBase="%imageBase%"
formsMessageListener="%formsMessageListener%"
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file **formsweb.cfg**, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the **basejini.htm** file and store it at another location. In the servlet configuration file, edit the **baseHTMLJinitiator** parameter to point to the new file.

URL to Record

If you do not have access to the startup HTML file, you can still have Oracle NCA record object names instead of its object ID by modifying the URL to record. The following solution only works if the startup HTML file does not reference another file while loading.

For this solution, you add **"?record=names"** after the URL in the Start Recording dialog box, after the URL name to record.



How to Launch Oracle Applications via the Personal Home Page

When launching Oracle Forms applications (versions 6i and higher) by logging in through the **Personal Home Page**, you must set several system profile options at the user level. It is desirable to pass such variables at the user level, and not at the site level, where it will affect all users.

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.

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for Oracle NCA protocol scripts.

Testing Secure Oracle NCA Applications

- ▶ In the Port Mapping node of the Recording Options dialog box, delete any existing entries for port 443 and create a new entry for the Oracle server name:

Service ID: HTTP

Target Server: Oracle Forms Server IP address or long host name

Target Port: 443

Connection Type: SSL

SSL Version: Active SSL version. If in doubt, select SSL 2/3.

Note that the **Service ID** is **HTTP** and not **NCA**.

For more information, see "Network Port Mapping Node" on page 388.

- ▶ If you encounter problems when replaying an NCA HTTPS script during the `nca_connect_server` command, insert the following function at the beginning of the script.

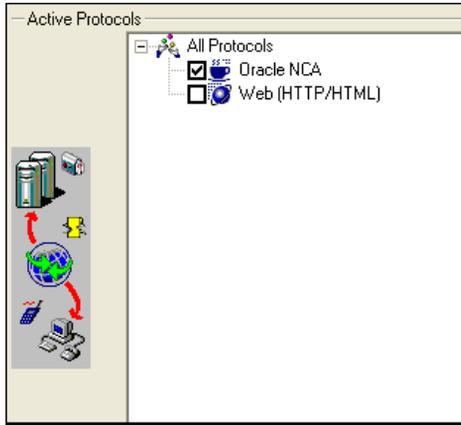
```
web_set_sockets_option("SSL_VERSION","3");
```

Testing Servlets and other Oracle NCA Applications

Certain NCA sessions use servlets:

- ▶ the Forms Listener servlet
- ▶ applications or modules that use both NCA and HTTP communications, such as the Oracle Configurator
- ▶ the initializing of the NCA application (downloading the applet, jar, and gif files)

When recording servlets, you must record both Oracle NCA and Web functions. You can do this by using the Oracle Apps Ili protocol or creating an Oracle NCA multi-protocol script. Alternatively, if you created a single protocol script for Oracle NCA, open the **General >Protocols** node in the Recording Options, and enable the Web protocol. Then you can begin recording.



If you are unsure whether your application uses servlets, you can check the **default.cfg** file in the script directory after recording a script. Locate the entry "**UseServletMode=**"

If the value is 1 or 2, then servlets are being used and you must enable HTTP recording in addition to Oracle NCA.

If you already recorded a script, you can regenerate the code automatically to include the Web functions without having to re-record. Select **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 (unless you are working with Forms Server 4.5). In systems where the standard port mapping are reserved by other applications, you may need to modify the Port Mapping settings, depending on the recording mode.

You can determine the recording mode in one of the following ways:

- When using the NCA application, open the Java Console.

```
proxyHost=null
proxyPort=0
connectMode=HTTP
Forms Applet version is: 60812
```

The **connectMode** entry indicates **HTTP**, **HTTPS**, or **socket**.

- After recording an NCA session, open the **default.cfg** file in the **Vuser** directory and check the value of the **UseHttpConnectMode** entry.

```
[HttpConnectMode]
UseHttpConnectMode= 2
// 0 = socket 1 = http 2 = https
```

When defining a new port mapping in the Server Entry dialog box, use a **Service ID** of HTTP for HTTP or HTTPS modes. For Socket mode, use a **Service ID** of NCA.

For more information about Port Mapping settings, see Chapter 10, "Network Port Mapping Node."

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**).

28

RDP Protocol

This chapter includes:

Concepts

- ▶ RDP Protocol Overview on page 786
- ▶ RDP Recording Tips on page 786
- ▶ Working with Clipboard Data on page 787
- ▶ Image Synchronization Overview on page 790
- ▶ Image Synchronization Tips on page 791
- ▶ Image Synchronization - Shifted Coordinates on page 792
- ▶ RDP Agent (Agent for Microsoft Terminal Server) Overview on page 793

Tasks

- ▶ How to Install / Uninstall the RDP Agent on page 796

Reference

- ▶ RDP User Interface on page 798

Troubleshooting and Limitations on page 801

Concepts

RDP Protocol Overview

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.

RDP 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.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Script Sections" on page 34.

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**.

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);
```

Image Synchronization Overview

An RDP session executes remotely. All keyboard and mouse handling is done on the server, and it is the server that reacts to them. For example, when you double-click an application on the desktop, it is the server that realizes a double-click took place and that the application must be loaded and displayed.

When an RDP client connects to a server, it does two things:

- ▶ It sends the server coordinates of actions. For example, 'clicked the left mouse button at coordinates (100, 100) on the screen'.
- ▶ It receives images from the server showing the current status of the screen after the action took place

The RDP client (and therefore, LoadRunner) does not know that the screen contains windows, buttons, icons, or other objects. It only knows the screen contains an image and at what coordinates the user performed the action. To allow the server to correctly interpret the actions, you set synchronization points within the script. These points instruct the script to wait until the screen on the server matches the stored screen before continuing.

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.

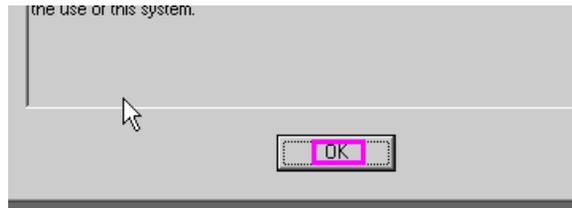
Image Synchronization Tips

Use the following guidelines for effective image synchronization:

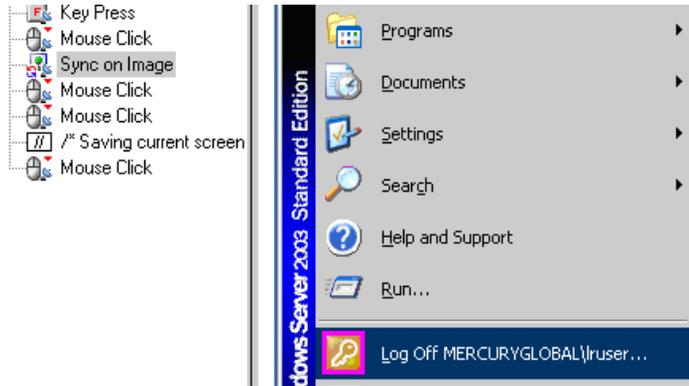
Synchronize on Smallest Significant Area

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.

Image Synchronization - 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.

RDP Agent (Agent for Microsoft Terminal Server) **Overview**

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides enhancements to the normal RDP functionality. It is provided in the product's DVD and you can install it on any RDP server. The agent provides you with more intuitive and readable scripts, built-in synchronization, and detailed information about relevant objects. In addition, 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.

The Agent for Microsoft Terminal Server provides the following enhancements to the normal RDP functionality:

Object Detail Recording

When the Agent for Microsoft Terminal Server is installed, VuGen can record specific information about the object that is being used instead of general information about the action. For example, VuGen generates **Sync Object Mouse Click** and **Sync Object Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows a double-mouse-click action recorded with and without the agent installation. Note that with the agent, VuGen generates `sync_object` functions for all of the mouse actions.

```
rdp_sync_object_mouse_double_click("StepDescription=Mouse Double Click on
Synchronized Object 1",
    "Snapshot=snapshot_12.inf",
    "WindowTitle=RDP2",
    "Attribute=TEXT",
    "Value=button1",
    "MouseX=100",
    "MouseY=71",
    "MouseButton=LEFT_BUTTON",
    RDP_LAST);

rdp_mouse_double_click("StepDescription=Mouse Double Click 1",
    "Snapshot=snapshot_2.inf",
    "MouseX=268",
    "MouseY=592",
    "MouseButton=LEFT_BUTTON",
    "Origin=Default",
    RDP_LAST);
```

Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When the agent is not active, you are limited to 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.

Tasks

How to Install / Uninstall the RDP 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).

This task includes the following steps:

- "Install the RDP Agent" on page 796
- "Uninstall the RDP Agent" on page 797

Install the RDP Agent

- 1** If your server requires administrator permissions to install software, log in as an administrator to the server.
- 2** If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

- 3** Locate the installation file, **Setup.exe**, on the LoadRunner DVD in the **Additional Components\ Agent for Microsoft Terminal Server** 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**.

Uninstall the RDP Agent

- 1** If your server requires administrator privileges to remove software, log in as an administrator to the server.
- 2** Select **Control Panel > Add/Remove Programs > HP Software Agent for Microsoft Terminal Server** and click **Change/Remove**.

Reference

RDP User Interface

This section includes (in alphabetical order):

- ▶ Failed Image Synchronization Dialog Box on page 799

Failed Image Synchronization Dialog Box

This dialog box opens when a synchronization fails. It enables you to decide to stop the script or continue despite the error.

To access	Opens automatically when a synchronization fails.
Important information	<p>THIS DIALOG BOX HAS A NUMBER OF DIFFERENT FORMS DEPENDING ON THE REASON FOR THE FAILED SYNCHRONIZATION:</p> <ul style="list-style-type: none"> ▶ 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. ▶ Raise Tolerance: The Failed Image Synchronization - Raise Tolerance dialog box opens when the script replay failed to find the exact image requested, but if the tolerance level for performing synchronization on images was relaxed, then it would have succeeded in finding the image. ▶ Lower Tolerance: The Failed Image Synchronization - Lower Tolerance dialog box opens when the script replay fails to meet the NotAppear or Change conditions. VuGen detected an image match where you expected it not to detect one. If the tolerance level was reduced, the recorded and replay images would not match, and the NotAppear or Change conditions would be met resulting in a successful replay. ▶ Non Specified: The Failed Image Synchronization dialog box opens when the script replay fails to meet any of the synchronization conditions such as NotAppear or Change. VuGen did not find another image at the original coordinates that could be appended to the script.
See also	"Image Synchronization Overview" on page 790

User interface elements are described below:

UI Elements (A-Z)	Description
Stop	Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
Continue	<p>This button performs different actions depending on the type of dialog box:</p> <p>Append Snapshot: 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. In this case, VuGen adds the new snapshot 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.</p> <p>Lower Tolerance: 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.</p> <p>Raise Tolerance: 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.</p> <p>Non-specified: Accept the mismatch, and do not make any changes in the script. Continue script execution despite the mismatch.</p> <p>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 tolerance for image synchronization setting in the Run-Time Settings > RDP > Synchronization node.</p>

Troubleshooting and Limitations

This section describes troubleshooting information for RDP scripts using the Agent for Microsoft Terminal Server.

Replay fails on `rdp_sync_object_mouse_click/double_click` steps

If the replay fails on specific `rdp_sync_object_mouse_click` or `rdp_sync_object_mouse_double_click` steps, there are workarounds to resolve the issue. We recommend that you try the workarounds in the order they are listed.

Workaround: Modify `RDPAgentCodeGen.cfg` file

The `RDPAgentCodeGen.cfg` file can configure VuGen to automatically create an `rdp_sync_on_image` and `rdp_mouse_click` step the next time the script is generated for each `rdp_sync_object_mouse_click/double_click` steps which occur within a given window. To do this, you specify the name of the window, update a variable which counts the total number of windows for which this process occurs, and regenerate the script.

To modify the `RDPAgentCodeGen.cfg` file:

- 1** Open the `RDPAgentCodeGen.cfg` file in the **Script Directory > data** directory.
- 2** Open the script in tree view and double-click the step that failed.
- 3** Copy the name of the window
- 4** In the `RDPAgentCodeGen.cfg` file, increase the value of `NumberOfTitles` by 1.
- 5** Add a line as follows:

```
WindowTitleX=<name of window>
```

where **X** is the new value of `NumberOfTitles`.

- 6** Regenerate the script.

Note: The **RDPAgentCodeGen.cfg** file can be used to automatically produce **rdp_sync_on_image** and **rdp_mouse_click** steps in a similar way for **rdp_sync_object_mouse_click/double_click** steps which are specified in different ways as well. Steps can be targeted based on the class attribute of the control. For more information, contact HP software support.

Workaround: Manually insert a new step

You can manually perform the workaround, by inserting an **rdp_sync_on_image** and **rdp_mouse_click** step for each step that fails. We do not recommend using this workaround because steps added in this way will be lost if the script is regenerated.

29

RTE Protocol

This chapter includes:

Concepts

- ▶ RTE Protocol Overview on page 804
- ▶ Working with Ericom Terminal Emulation on page 805
- ▶ Typing Input into a Terminal Emulator on page 807
- ▶ Generating Unique Device Names on page 810
- ▶ Setting the Field Demarcation Characters on page 811
- ▶ Reading Text from the Terminal Screen on page 812
- ▶ RTE Synchronization Overview on page 813
- ▶ Synchronizing Block-Mode (IBM) Terminals on page 814
- ▶ Synchronizing Character-Mode (VT) Terminals on page 818

Tasks

- ▶ How to Map Terminal Keys to PC Keyboard Keys on page 823
- ▶ How to Record RTE Scripts on page 824
- ▶ How to Implement Continue on Error on page 827

Concepts

RTE Protocol Overview

An RTE Vuser types character input into a terminal emulator, submits the data to a server, and then waits for the server to respond. For instance, suppose that you have a server that maintains customer information for a maintenance company. 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 "RTE Synchronization Overview" on page 813.

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.

For syntax and examples of the TE functions, see the *Online Function Reference* (**Help > Function Reference**).

An RTE Vuser emulates the actions of a real user. Human users use terminals or terminal emulators to operate application programs.



In the RTE Vuser environment, a Vuser replaces the human. The Vuser operates PowerTerm, a terminal emulator.



PowerTerm works like a standard terminal emulator, supporting common protocols such as IBM 3270 & 5250, VT100, VT220, and VT420-7.

Working with Ericom Terminal Emulation

VuGen supports record and replay with Ericom Terminal Emulators.

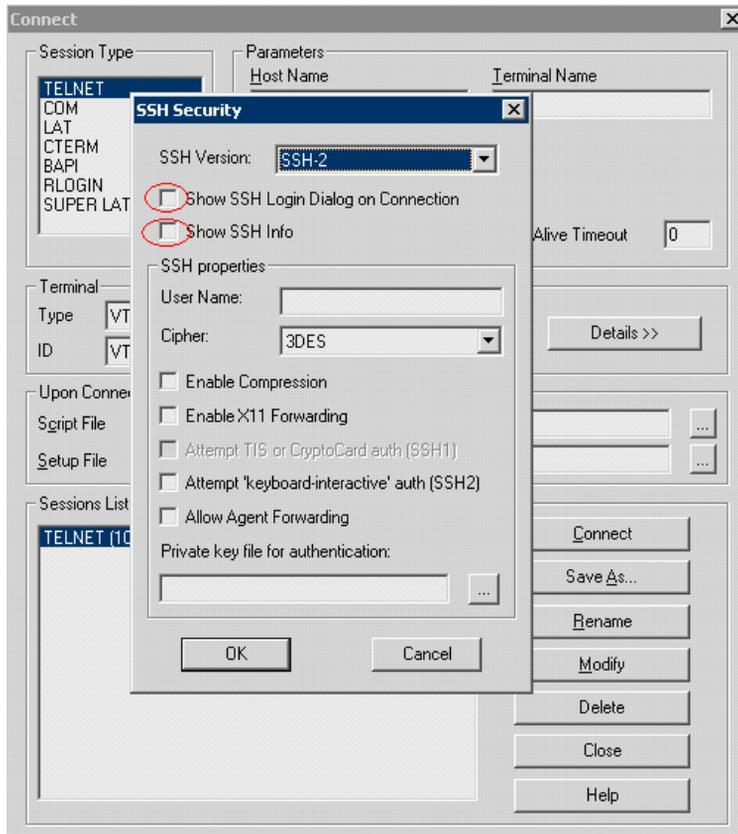
The Ericom support handles escape sequences during record and replay. Ericom's PowerTerm lets you map PC keys to custom escape sequences. For information about mapping, see the PowerTerm help.

When a user presses mapped keys while recording an Ericom VT session, VuGen generates **TE_send_text** functions instead of the standard **TE_type**. This allows the script to handle custom escape sequences in a single step. For more information, see the *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 pop-ups from being issued. If you enable these pop-ups, it may affect the replay. You can access the advanced security options by clicking the **Details** button.



Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to "type" character input into the PowerTerm terminal emulator:

- ▶ **TE_type** sends characters to the terminal emulator. When recording, the VuGen automatically generates **TE_type** functions for keyboard input to the terminal window. For details, see below.
- ▶ **TE_typing_style** determines the speed at which the Vuser types. You can manually define the typing style by inserting a **TE_typing_style** function into the Vuser script. For details, see "Setting the Typing Style" on page 809. Alternatively, you can set the typing style by using the run-time settings. For more information, see Chapter 11, "Run-Time Settings".

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 11, "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 11, "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**.

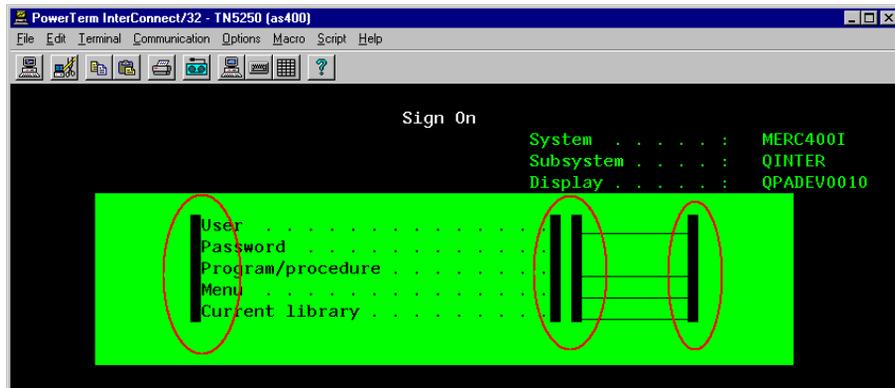
If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

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.

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**).

RTE Synchronization Overview

Depending on the system you are testing, you may need to synchronize the input that a Vuser sends to a terminal emulator with the subsequent responses from the server. When you synchronize input, you instruct the Vuser to suspend script execution and wait for a cue from the system, before the Vuser performs its next action. For instance, suppose that a human user wants to submit the following sequence of key strokes to a bank application:

- 1** Type 1 to select "Financial Information" from the menu of a bank application.
- 2** When the message "What information do you require?" appears, type 3 to select "Dow Jones Industrial Average" from the menu.
- 3** When the full report has been written to the screen, type 5 to exit the bank application.

In this example, the input to the bank application is synchronized because at each step, the human user waits for a visual cue before typing. This cue can be either the appearance of a particular message on the screen, or stability of all the information on the screen.

You can synchronize the input of a Vuser in the same way by using the TE-synchronization functions, **TE_wait_sync**, **TE_wait_text**, **TE_wait_silent**, and **TE_wait_cursor**. These functions effectively emulate a human user who types into a terminal window and then waits for the server to respond, before typing in the next command.

The **TE_wait_sync** function is used to synchronize block-mode (IBM) terminals only. The other TE-synchronization functions are used to synchronize character-mode (VT) terminals.

When you record an RTE Vuser script, VuGen can automatically generate and insert **TE_wait_sync**, **TE_wait_text**, and **TE_wait_cursor** statements into the script. You use VuGen's recording options to specify which synchronization functions VuGen should insert.

Note: Do not include any synchronization statements in the *Vuser_end* section of a Vuser script. Since a Vuser can be aborted at any time, you cannot predict when the *Vuser_end* section will be executed.

Synchronizing Block-Mode (IBM) Terminals

The **TE_wait_sync** function is used for synchronization RTE Vusers operating block-mode (IBM) terminals. Block-mode terminals display the "X SYSTEM" message to indicate that the system is in Input Inhibited mode. When a system is in the Input Inhibited mode no typing can take place because the terminal emulator is waiting for a transfer of data from the server.

When you record a script on a block-mode terminal, by default, VuGen generates and inserts a **TE_wait_sync** function into the script each time the "X SYSTEM" message appears. You use VuGen's recording options to specify whether or not VuGen should automatically insert **TE_wait_sync** functions.

When you run a Vuser script, the **TE_wait_sync** function checks if the system is in the X SYSTEM mode. If the system is in the X SYSTEM mode, the **TE_wait_sync** function suspends script execution. When the "X SYSTEM" message is removed from the screen, script execution continues.

Note: You can use the **TE_wait_sync** function only with IBM block-mode terminals emulators (5250 and 3270).

In general, the **TE_wait_sync** function provides adequate synchronization for all block-mode terminal emulators. However, if the **TE_wait_sync** function is ineffective in a particular situation, you can enhance the synchronization by including a **TE_wait_text** function. For more information on the **TE_wait_text** function, see "Waiting for Text to Appear on the Screen" on page 820, and the *Online Function Reference (Help > Function Reference)*.

In the following script segment, the Vuser logs on with the user name "QUSER" and the password "HPLAB". The Vuser then presses Enter to submit the login details to the server. The terminal emulator displays the X SYSTEM message while the system waits for the server to respond.

The **TE_wait_sync** statement causes the Vuser to wait until the server has responded to the login request, that is, for the X SYSTEM message to be removed—before executing the next line of the script.

```
TE_type("QUSER");
lr_think_time(2);
TE_type("<kTab>HPLAB");
lr_think_time(3);
TE_type("<kEnter>");
TE_wait_sync();
....
```

When a **TE_wait_sync** function suspends the execution of a script while an X SYSTEM message is displayed, the Vuser continues to monitor the system—waiting for the X SYSTEM message to disappear. If the X SYSTEM message does not disappear before the synchronization timeout expires, the **TE_wait_sync** function returns an error code. The default timeout is 60 seconds.

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**).

Tasks

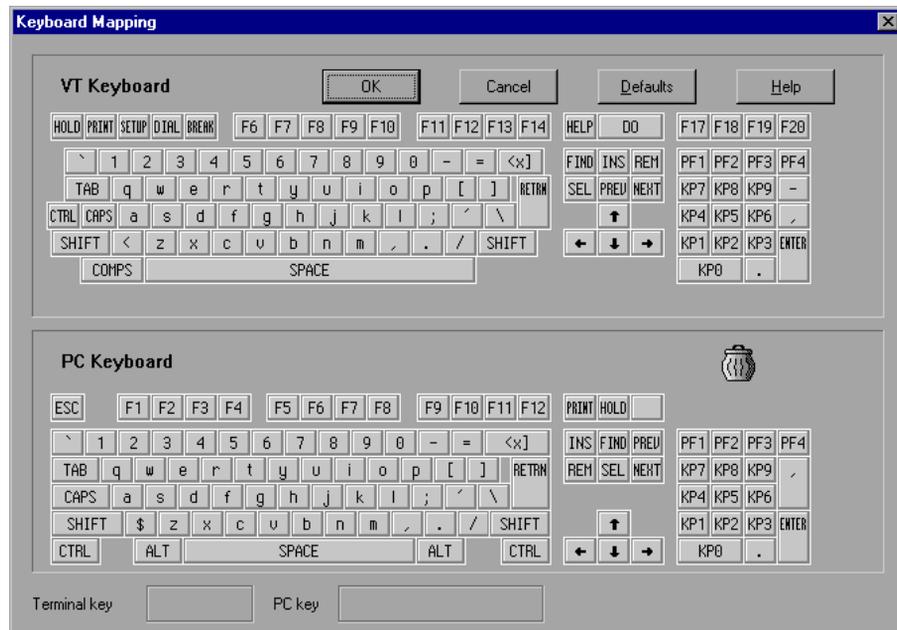
How to Map Terminal Keys to PC Keyboard Keys

Because you are using a terminal emulator, you will be using a PC keyboard in place of a terminal keyboard. Many keys that are found on the terminal keyboard are not available on a PC keyboard. Examples of such keys are HELP, 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**.

How to Record RTE 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.

This task describes how to record RTE scripts. This procedure differs from the general recording procedure in Chapter 3, "Recording".

This task includes the following steps:

- "Record the terminal setup and connection" on page 824
- "Record typical user actions" on page 827
- "Record the log-off procedure" on page 827

1 Record the terminal setup and connection

- a Open an existing RTE Vuser script, or create a new one.
- b In the **Sections** box, select the **vuser_init** section to insert the recorded statements.
- c In the Vuser script, place the cursor at the location where you want to begin recording.
- d Click the **Start Record** button. The PowerTerm main window opens.
- e From the PowerTerm menu bar, select **Terminal > Setup** to display the Terminal Setup dialog box.



- f** Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.
-

Note: Select an IBM terminal type to connect to an AS/400 machine or an IBM mainframe; select a VT terminal type to connect to a UNIX workstation.

- g** Select **Communication > Connect** to display the Connect dialog box.
 - h** Under **Session Type**, select the type of communication to use.
 - i** Under **Parameters**, specify the required options. The available parameters vary depending on the type of session that you select. For details on the parameters, click **Help**.
-

Tip: Click **Save As** to save the parameter-sets for re-use in the future. The parameter-sets that you save are displayed in the Sessions List box.

- j Click **Connect**. PowerTerm connects to the specified system, and VuGen inserts a **TE_connect** function into the script, at the insertion point. The **TE_connect** statement has the following form:

```

/* *** The terminal type is VT 100. */
TE_connect(
    "comm-type = telnet;"
    "host-name = alfa;"
    "telnet-port = 992;"
    "terminal-id = ;"
    "set-window-size = true;"
    "security-type = ssl;"
    "ssl-type = tls1;"
    "terminal-type = vt100;"
    "terminal-model = vt100;"
    "login-command-file = ;"
    "terminal-setup-file = ;"
    , 60000);
if (TE_errno != TE_SUCCESS)
    return -1;

```

The inserted **TE_connect** statement is followed by an if statement that checks whether or not the **TE_connect** function succeeds during replay.

Note: Do not record more than one connection to a server (**TE_connect**) in a Vuser script.

2 Record typical user actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the **Actions** section of the Vuser script. Only the **Actions** section of a Vuser script is repeated when you run multiple iterations of the script.

When recording a session, VuGen records the text strokes and not the text. Therefore, it is not recommended that you copy and paste commands into the PowerTerm window—instead, type them in directly.

- a Select the **Actions** section in the **Section** box.
- b Proceed to perform typical user actions in the terminal emulator. VuGen generates the appropriate statements, and inserts them into the Vuser script while you type. If necessary, you can edit the recorded statements while you record the script.

3 Record the log-off procedure

- a Make sure that you have performed and recorded the typical user actions as described in the previous section.
- b In the VuGen main window, click **vuser_end** in the **Section** box.
- c Perform the log off procedure. VuGen records the procedure into the **vuser_end** section of the script.
- d Click **Stop Recording on** the Recording toolbar. The main VuGen window displays all the recorded statements.
- e Click **Save** to save the recorded session. After recording a script, you can manually edit it in VuGen's main window.



How to Implement Continue on Error

To configure the Continue on Error functionality in RTE Scripts:

- To continue running the script on error, insert the following function:
`TE_setvar(TE_IGNORE_ERRORS, 1)`
- To restore the default behavior of failing the script on error, insert the following function:

```
TE_setvar(TE_IGNORE_ERRORS, 0)
```

30

SAP Protocols

This chapter includes:

Concepts

- ▶ Selecting a SAP Protocol Type on page 830
- ▶ SAPGUI Protocol on page 831
- ▶ SAP Web Protocol on page 835
- ▶ SAP (Click and Script) Protocol on page 836
- ▶ Replaying SAPGUI Optional Windows on page 839

Tasks

- ▶ How to Configure the SAP Environment on page 840
- ▶ How to Record SAPGUI Scripts on page 847
- ▶ How to Replay SAPGUI Script on page 849
- ▶ How to Run SAPGUI Scripts in a Scenario on page 849
- ▶ How to Enhance SAPGUI Scripts on page 851

Reference

- ▶ Additional SAP Resources on page 859

Troubleshooting and Limitations on page 859

Concepts

Selecting a SAP Protocol Type

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) or SAP-Web protocol.

To record a SAPGUI session that uses browser controls, create a multi-protocol Vuser script with the SAPGUI and SAP-Web protocols. This allows VuGen to record Web-specific functions when encountering the browser controls. This will not work if you attempt to combine SAPGUI and Web protocols.

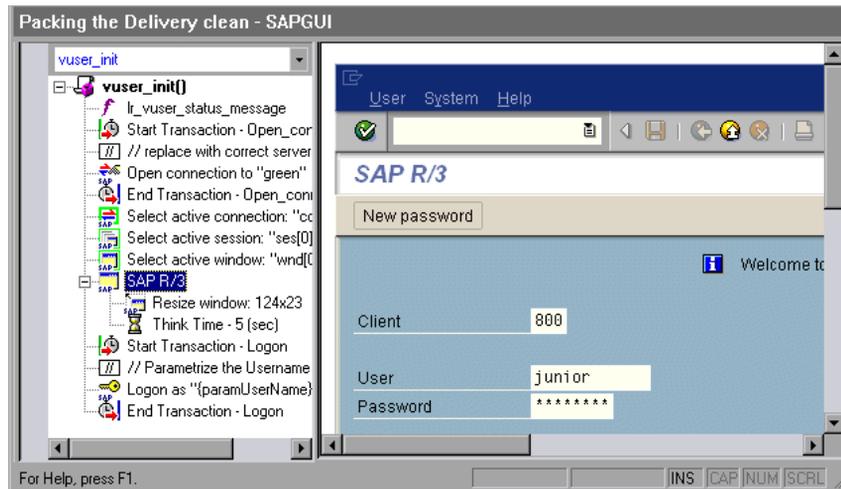
Before recording a session, verify that your modules and client interfaces are supported by VuGen. The following table describes the SAP client modules for SAP Business applications and the relevant tools:

SAP module	VuGen support
SAP Web Client or mySAP.com.	Use the SAP-Web protocol.
SAPGUI for Windows.	Use the SAPGUI protocol. This also supports APO module recording (requires patch level 24 for APO 3.0 for SAP 6.20).
SAPGUI for Windows and a web browser.	Use the SAP (Click and Script) protocol.
SAPGUI for Java.	This client is not supported.

SAPGUI Protocol

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.

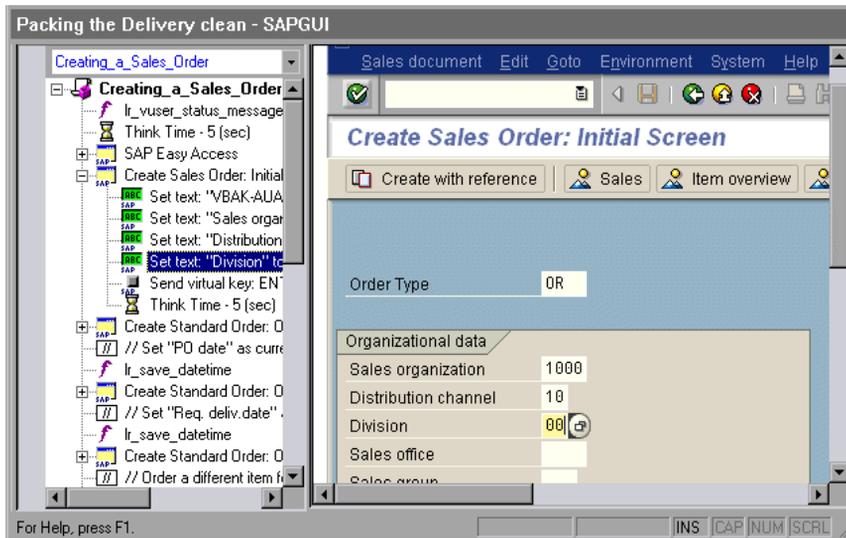
The following example shows a typical recording of a SAPGUI client. The first section, **vuser_init**, contains the opening of a connection and a logon.



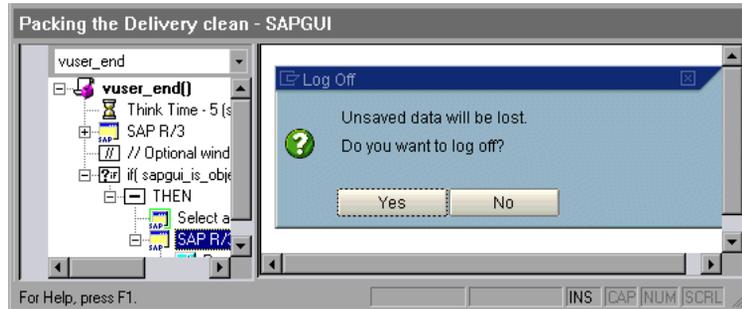
Note that the Open Connection step uses one of the connection names in the SAP Logon **Descriptions** list. If the specified connection name is not in the list, the Vuser looks for a server with that name.



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=7pt5civ1p3m2&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);
```

SAP Web Protocol

The SAP-Web Vuser script typically contains several SAP transactions which make up a business process. The business process consists of functions that emulate user actions. For information about these functions, see the Web functions in the *Online Function Reference* (**Help > Function Reference**).

Example:

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.com:80/Action/26011[header]", 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[header]", ENDITEM,
            ...
            LAST);
```

The following section illustrates a SAP Web and SAPGUI 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/
dummy?PASS_PARAMS=YES&dummyComp=dummy&Tcode=VA01&draggable=0&C
ompFName=VA01&Style=sap_mango_polarwind",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.hplab.com/sapportal",
        "Snapshot=t9.inf",
        "Mode=HTML",
        LAST);

sapgui_open_connection_ex("/H/Protector/S/3200 /WP",
    "",
    "con[0]");

sapgui_select_active_connection("con[0]");

sapgui_select_active_session("ses[0]");

/*Before running script, enter password in place of asterisks in logon function*/

sapgui_logon("JUNIOR{UserNumber}",
    "ides",
    "800",
    "EN",
    BEGIN_OPTIONAL,
    "AdditionalInfo=sapgui102",
    END_OPTIONAL);

```

SAP (Click and Script) Protocol

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.

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported SAP objects, VuGen generates a function with an `sap_xxx` prefix.

Example:

In the following example, a user selected the **User Profile** tab. VuGen generated a **sap_portal** function.

```
web_browser("Close_2",
  "Snapshot=t7.inf",
  DESCRIPTION,
  "Ordinal=2",
  ACTION,
  "UserAction=Close",
  LAST);

lr_think_time(7);

web_text_link("Personalize",
  "Snapshot=t8.inf",
  DESCRIPTION,
  "Text=Personalize",
  ACTION,
  "UserAction=Click",
  LAST);

lr_think_time(6);

sap_portal("Sap Portal_2",
  "Snapshot=t9.inf",
  DESCRIPTION,
  "BrowserOrdinal=2",
  ACTION,
  "DetailedNavigation=User Profile",
  LAST);
```

Note: When you record 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.

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.

Tasks

How to Configure the SAP Environment

This task describes configure and verify the SAP environment for use with VuGen.

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.

This task includes the following steps:

- ▶ "Checking the SAPGUI for Windows client patch level" on page 840
- ▶ "Check the kernel patch level" on page 841
- ▶ "Check the R/3 support packages" on page 842
- ▶ "Enable scripting on the SAP application server" on page 844
- ▶ "Enable scripting on SAPGUI 6.20 client" on page 846

Checking the SAPGUI for Windows client 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** Open 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.

Check the kernel patch level

- 1** Log in to the SAP system
- 2** Select **System > Status**
- 3** Click the **Other kernel information button**.
- 4** In the **Kernel Information** section, check the value of the **Sup. Pkg. lvl.**



The level must be greater than the level listed in the following chart depending on the SAP version you are using.

Software Component	SAP Release	Kernel Patch Level
SAP_APPL	31I	Kernel 3.1I level 650
SAP_APPL	40B	Kernel 4.0B level 903
SAP_APPL	45B	Kernel 4.5B level 753
SAP_BASIS	46B	Kernel 4.6D level 948
SAP_BASIS	46C	Kernel 4.6D level 948
SAP_BASIS	46D	Kernel 4.6D level 948
SAP_BASIS	610	Kernel 6.10 level 360

Check the R/3 support packages

- 1 Log on to the SAP system and run the SPAM transaction.
- 2 In the **Directory** section, select **All Support Packages**, and click the **Display** button.
- 3 Verify that the correct package is installed for your version of SAP according to the table below.

Software Component	Release	Package Name
SAP_APPL	31I	SAPKH31I96
SAP_APPL	40B	SAPKH40B71
SAP_APPL	45B	SAPKH45B49
SAP_BASIS	46B	SAPKB46B37
SAP_BASIS	46C	SAPKB46C29
SAP_BASIS	46D	SAPKB46D17
SAP_BASIS	610	SAPKB61012

If the correct version is installed, a green circle appears in the Status column.

The screenshot shows the SAP OCS package directory for 'All packages'. The table lists various support packages for SAP Basis 4.6C, with their status and import dates. The status column contains green circles, indicating that all packages are installed. The packages listed are:

Package Name	Short description	Status	Import status
SAP_BASIS_46C	SAP Basis Component	●●●	
SAPKB46C01	Basis Support Package 01 for 4.6C (Extra L...	●●●	imported at 22.06.2000, 17:06:15
SAPKB46C02	Basis Support Package 02 for 4.6C	●●●	imported at 25.07.2000, 21:28:19
SAPKB46C03	Basis Support Package 03 for 4.6C	●●●	imported at 25.07.2000, 21:28:20
SAPKB46C04	Basis Support Package 04 for 4.6C	●●●	imported at 06.09.2000, 08:50:41
SAPKB46C05	Basis Support Package 05 for 4.6C	●●●	imported at 06.09.2000, 08:50:42
SAPKB46C06	Basis Support Package 06 for 4.6C (Extra ...	●●●	imported at 06.09.2000, 08:50:43
SAPKB46C07	Basis Support Package 07 for 4.6C (Extra ...	●●●	imported at 13.11.2000, 12:47:00
SAPKB46C08	Basis Support Package 08 for 4.6C	●●●	imported at 13.11.2000, 12:47:01
SAPKB46C09	Basis Support Package 09 for 4.6C	●●●	imported at 13.11.2000, 12:47:02
SAPKB46C10	Basis Support Package 10 for 4.6C	●●●	imported at 29.01.2001, 12:54:15
SAPKB46C11	Basis Support Package 11 for 4.6C	●●●	imported at 29.01.2001, 12:54:16
SAPKB46C12	Basis Support Package 12 for 4.6C	●●●	imported at 29.01.2001, 12:54:17
SAPKB46C13	Basis Support Package 13 for 4.6C	●●●	imported at 29.01.2001, 12:54:18
SAPKB46C14	Basis Support Package 14 for 4.6C	●●●	imported at 15.05.2002, 12:28:32
SAPKB46C15	Basis Support Package 15 for 4.6C	●●●	imported at 15.05.2002, 12:28:33
SAPKB46C16	Basis Support Package 16 for 4.6C	●●●	imported at 15.05.2002, 12:28:34
SAPKB46C17	Basis Support Package 17 for 4.6C	●●●	imported at 15.05.2002, 12:28:35
SAPKB46C18	Basis Support Package 18 for 4.6C	●●●	imported at 15.05.2002, 12:28:36
SAPKB46C19	Basis Support Package 19 for 4.6C	●●●	imported at 15.05.2002, 12:28:37
SAPKB46C20	Basis Support Package 20 for 4.6C	●●●	imported at 15.05.2002, 12:28:38
SAPKB46C21	Basis Support Package 21 for 4.6C	●●●	imported at 15.05.2002, 12:28:39
SAPKB46C22	Basis Support Package 22 for 4.6C	●●●	imported at 15.05.2002, 12:28:40
SAPKB46C23	Basis Support Package 23 for 4.6C	●●●	imported at 15.05.2002, 12:28:41
SAPKB46C24	Basis Support Package 24 for 4.6C	●●●	imported at 15.05.2002, 12:28:42
SAPKB46C25	Basis Support Package 25 for 4.6C	●●●	imported at 15.05.2002, 12:28:43
SAPKB46C26	Basis Support Package 26 for 4.6C	●●●	imported at 15.05.2002, 12:28:44
SAPKB46C27	Basis Support Package 27 for 4.6C	●●●	imported at 15.05.2002, 12:28:45
SAPKB46C28	Basis Support Package 28 for 4.6C	●●●	imported at 15.05.2002, 12:28:46
SAPKB46C29	Basis Support Package 29 for 4.6C	●●●	imported at 15.05.2002, 12:28:47

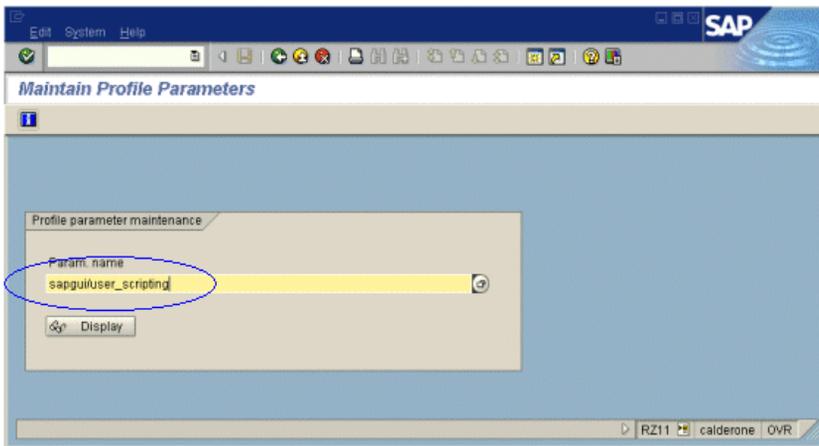
If you do not have the OCS package installed, download it from the www.sap.com Web site and install it. For more information, see the SAP OSS note #480149.

Enable scripting on the SAP application server

A user with administrative permissions enables scripting by setting the **sapgui/user_scripting** profile parameter to **TRUE** on the application server. To enable scripting for all users, set this parameter on all application servers. To enable scripting for a specific group of users, only set the parameter on application servers with the desired access restrictions.

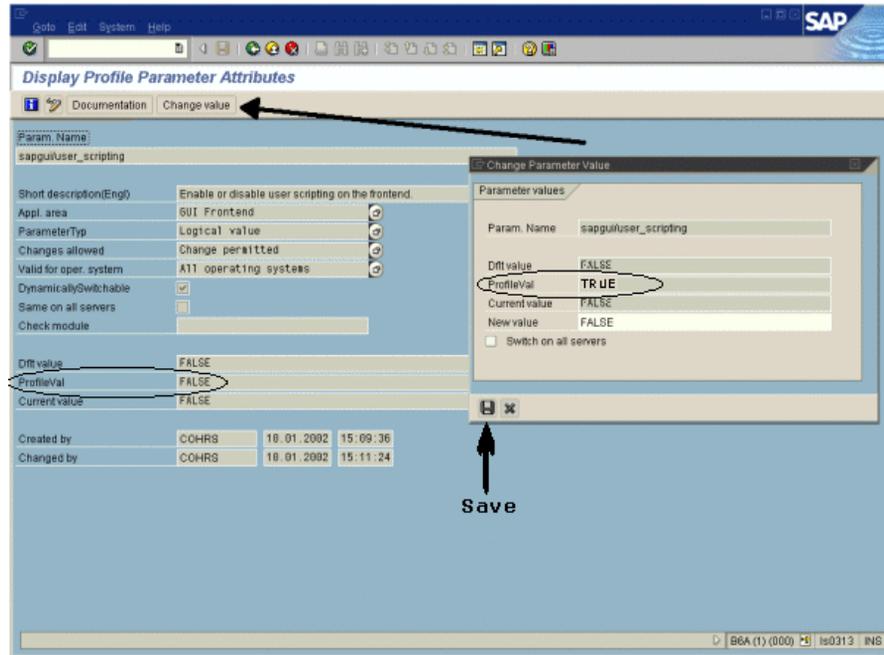
To change the profile parameter:

- 1 Open transaction **rz11**. Specify the parameter name **sapgui/user_scripting** and click **Display**. The Display Profile Parameter Attributes window opens.



If **Parameter name is unknown** appears in the status bar, this indicates that you are missing the current Support Package. Import the Support Package that corresponds to the SAP BASIS and kernel versions of the application server, as described in the steps above.

- 2 If **Profile Val** is FALSE, you need to modify its value. Click the **Change value** button in the toolbar. The Change Parameter Value window opens. Enter TRUE in the **ProfileVal** box and click the **Save** button.



When you save the change, the window closes and **ProfileVal** is set to TRUE.

- 3 Restart the application server, since this change only takes effect when you log onto the system.

If the updated **ProfileVal** did not change, even after restarting the server, then the kernel of the application server is outdated. Import the required kernel patch, as specified in the steps above.

Note that the Profile Value may be dynamically activated in the following kernel versions, using transaction rz11, without having to restart the application server.

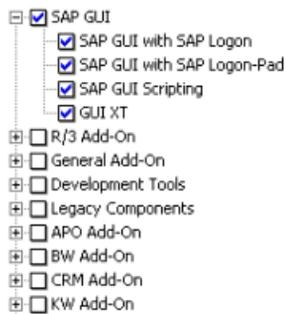
Release	Kernel Version	Patch Level
4.6B, 4.6C, 4.6D	4.6D	972
6.10	6.10	391
6.20	all versions	all levels

Enable scripting on 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:

- 1 During installation.** While installing the SAPGUI client, enable the **SAP GUI Scripting** option.



- 2 After installation.** Suppress warning messages. Open the Options dialog box in the SAPGUI client. Select the **Scripting** tab and clear the following options:

- **Notify when a script attaches to a running GUI**

► **Notify when a script opens a connection**

You can also prevent these messages from popping up by setting the values **WarnOnAttach** and **WarnOnConnection** in the following registry key to 0:

```
HKCU\SOFTWARE\SAP\SAPGUI Front\SAP Frontend Server\Security.
```

How to Record SAPGUI Scripts

The following steps describe some prerequisites to recording a SAPGUI script.

- "Close SAPLogon application when recording with multi" on page 847
- "Use modal dialog boxes for F1 help" on page 847
- "Use modal dialog boxes for F4 help." on page 847

Close SAPLogon application when recording with multi

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 help

Instruct the SAPGUI client to open the F1 help in a modal dialog box as follows:

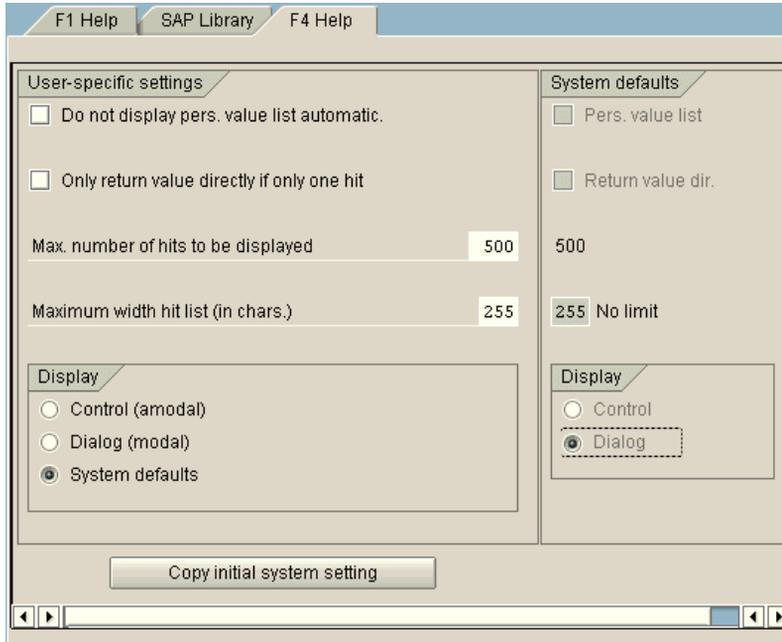
- 1** Select **Help > Settings**.
- 2** Click the **F1 Help** tab.
- 3** Select **in modal dialog box** in the Display section.

Use modal dialog boxes for F4 help.

Note: This procedure can only be performed by the administrator.

Instruct the SAPGUI client to open the F4 help in a modal dialog box as follows:

- 1 Make sure that all users have logged off from the server.
- 2 Select **Help > Settings**. Click the **F4 Help** tab.



- 3 In the Display section, select **System defaults**.
- 4 In the Display portion of the System defaults section, select **Dialog**.
- 5 Save the changes by clicking **Copy initial system setting**.
- 6 Verify that the status bar displays the message **Data was saved**.
- 7 Close the session and restart the service through the SAP Management Console.

How to Replay SAPGUI Script

The following steps describe prerequisites to replaying SAPGUI scripts.

- "Replace encrypted password" on page 849
- "Display SAPGUI user interface during replay (optional)" on page 849

Replace encrypted password

Replace the encrypted password in the `sapgui_logon` function generated during recording, with the real password. It is the second argument of the function, after the following user name

```
sapgui_logon("user", "pswd", "800", "EN");
```

For additional security, you can encrypt the password within the code. Select the password text (the actual text, not *****) and select **Encrypt string** from the right-click menu. VuGen inserts an `lr_decrypt` function at the location of the password as follows:

```
sapgui_logon("user", lr_decrypt("3ea037b758"), "800", "EN");
```

Display SAPGUI user interface during replay (optional)

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. Select **Vuser > Run-Time Settings > SAPGUI > General** node and select **Show SAP Client During Replay**. During a load scenario, disable this option, since it uses a large amount of system resources in displaying the UI for multiple Vusers.

How to Run SAPGUI Scripts in a Scenario

The following steps describe tips for running SAPGUI scripts in a scenario.

- "LoadRunner controller settings" on page 850
- "Make sure the agent is running in process mode" on page 850

LoadRunner controller settings

When working with a LoadRunner scenario, set the following values when running your script in a load test configuration:

- ▶ **Ramp-up.** One by one (to insure proper logon) in the Scheduler.
- ▶ **Think time.** Random think time in the Run-Time settings.
- ▶ **Users per load generator.** 50 Vusers for machine with 512 MB of memory in the Load Generators dialog box.

Make sure the agent is running in process mode

Make sure that the LoadRunner (or Performance Center) Remote Agent is running in Process mode. Service mode is not supported.

To check this, move your mouse over the agent's icon in the Windows task bar area, and read the description. If the description reads LoadRunner Agent Service, it is running as a service.



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 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.

How to Enhance SAPGUI Scripts

The following steps describe additional options that are available to enhance SAPGUI scripts.

- "Record at the cursor" on page 852
- "Insert Steps Interactively into a SAPGUI Script" on page 853
- "Add verification functions" on page 855
- "Retrieve information" on page 856
- "Save date information" on page 858

Record at the cursor

VuGen also allows you to record actions into an existing script by either inserting new actions or replacing existing actions. You may decide to record into an existing script for several reasons:

- ▶ You made a mistake in the actions that you performed during recording.
- ▶ Your actions were correct, but you need to add additional information such as the handling of popup windows. For example the SAP server may issue an inventory warning, which did not apply during the recording session.

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.**
 - a** **Insert steps into action** inserts the newly recorded steps at the cursor without overwriting any existing steps. The new segment is enclosed with comments indicating the beginning and end of the added section. This option is ideal for handling occasional popup windows that were not present during the recording
 - b** **Overwrite the rest of the script** replaces all steps from the point of the cursor onward. This option overwrites the remainder of the current Action and deletes all other Actions. It does not affect the **vuser_init** or **vuser_end** sections.
- 4** Click **OK**. VuGen replays the script until the point of the cursor.
- 5** Wait for the Recording floating toolbar to open to begin performing actions in the SAPGUI client, switching between sections and actions as required.
- 6** Click the **Stop** button on the floating toolbar to end the recording session.



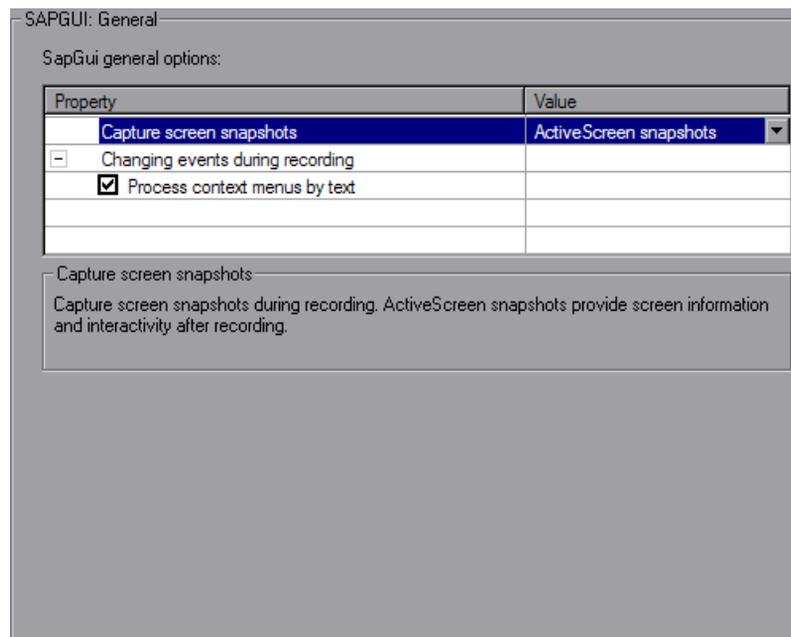
Insert Steps Interactively into a SAPGUI Script

After recording, you can manually add steps to the script in either Script view and Tree View. 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 Node).

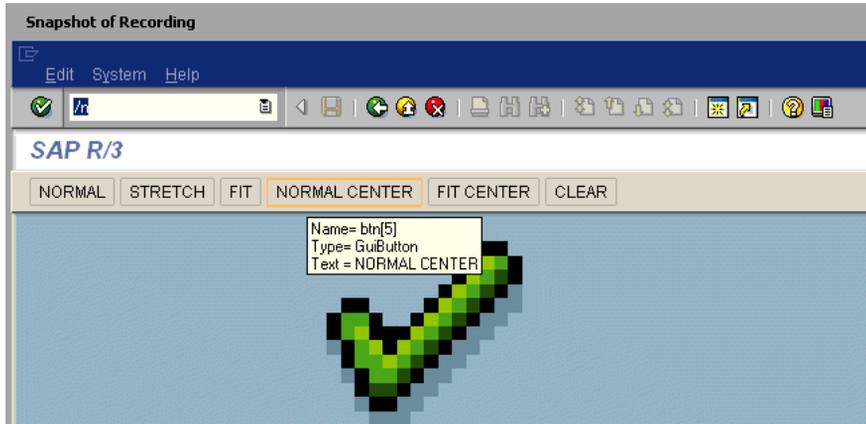
To insert a step interactively for a specific object:

- 1 Verify that you recorded the script when Active Screen snapshots were selected in the SAPGUI General node of the Recording Options (enabled by default).

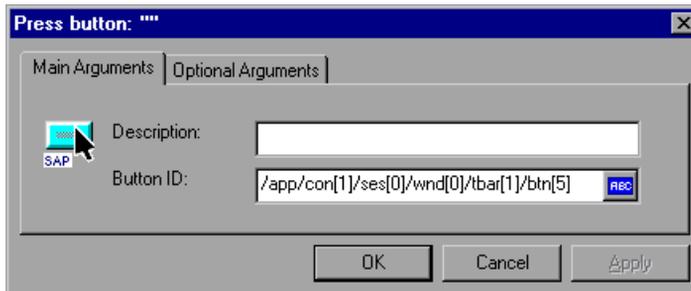


- 2 Click within the Snapshot window.

- 3 Move the mouse over the object for which you want to add a function. Make sure that VuGen recognizes the object and encloses it with a box.



- 4 Select **Insert New Step** from the right-click menu. The Insert Step box opens.
- 5 Select a step from the menu. The step's Properties dialog box opens, with the Control ID of the object when relevant. For example, if you add a **Press Button** step, for the NORMAL CENTER button as shown above, the Properties box displays the following ID:



- 6 Enter a name for the object in the **Description** box. Click **OK**. VuGen inserts the new step after the selected step.

Note: You can get the Control ID of the object for the purpose of pasting it into a specific location. To do this, select Copy **Control ID** from the right-click menu. You can past it into a Properties box or directly into the code from the Script view.

Add verification functions

When working with optional or dynamic windows or frames, you can use verification functions to determine if the window or object is available. An optional window is a window that does not consistently open during the SAP session. This function allow the Vuser script to continue running even if an optional window opens or an exception occurs.

The first example checks if a window is available. If the window is available, the Vuser closes it before continuing.

```
if (!sapgui_is_object_available("wnd[1]"))
    sapgui_call_method("{ButtonID}",
        "press",
        LAST,
        AdditionalInfo=info1011");
sapgui_press_button(.....)
```

The next example illustrates a dynamic object in the ME51N transaction. The Document overview frame is optional, and can be opened/closed by the **Document overview on/off** button.

The code checks the text on the Document overview button. If the text on the button shows Document overview on, we click the button to close the Document overview frame.

```

if(sapgui_is_object_available("tbar[1]/btn[9]"))
{
    sapgui_get_text("Document overview on/off button",
        "tbar[1]/btn[9]",
        "paramButtonText",
        LAST);

    if(0 == strcmp("Document overview off", lr_eval_string("{paramButtonText}")))
        sapgui_press_button("Document overview off",
            "tbar[1]/btn[9]",
            BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui1013",
            END_OPTIONAL);
}

```

Retrieve information

When working with SAGUI Vusers, you can retrieve the current value of a 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.

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"
```

Save date information

When creating scripts that use dates, your script may not run properly. For example, if you record the script on June 2, and replay it on June 3, the date fields will be incorrect. Therefore, you need to save the date to a parameter during text execution, and use the stored value as input for other date fields. To save the current date or time during script execution, use the **lr_save_datetime** function. Insert this function before the function requiring the date information. Note that the format of the date is specific to your locale. Use the relevant format within the **lr_save_datetime** function. For example, for month.day.year, specify "%m.%d.%Y".

In the following example, **lr_save_datetime** saves the current date. The **sapgui_set_text** function uses this value to set the delivery date for two days later.

```
lr_save_datetime("%d.%m.%Y", DATE_NOW + (2 * ONE_DAY),
  "paramDateTodayPlus2");

sapgui_set_text("Req. deliv.date",
  "{paramDateTodayPlus2}",
  "usr/ctxtRV45A-KETDAT",
  BEGIN_OPTIONAL,
  "AdditionalInfo=sapgui1025",
  END_OPTIONAL);
```

Reference

Additional SAP Resources

For more information, see the SAP website at www.sap.com or one of the following locations:

- ▶ **SAP Notes** - <https://websmp103.sap-ag.de/notes>

Note #480149: New profile parameter for user scripting on the front end

Note #587202: Drag & Drop is a known limitation of the SAPGUI interface

- ▶ **SAP Patches** - <https://websmp104.sap-ag.de/patches>

SAP GUI for Windows - SAPGUI 6.20 Patch (the lowest allowed level is 32)

Troubleshooting and Limitations

This section describes troubleshooting and limitations for SAPGUI, SAP Web, and SAP (Click and Script) protocols.

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 "How to Replay SAPGUI Script" on page 849.

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 "How to Configure the SAP Environment" on page 840.
- 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 "Add verification functions" on page 855.

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.

Question 8: When I go to the Auto Logon node of the Recording Options, why is the list of server names empty?

Answer: This sometimes occurs when using SAPGUI Client 7.20. To resolve this issue, copy the **saplogon.ini** file from **%APPDATA%\SAP\Common** where **%APPDATA%** stands for the environment variable specifying the Application Data directory located directly below the user profile directory. Paste the file to the **%WINDIR%** directory (C:\Windows).

31

Siebel Web Protocol

This chapter includes:

Concepts

- ▶ Siebel Web Protocol Overview on page 864
- ▶ Siebel Web Recording Options and Run-Time Settings on page 864

Tasks

- ▶ How to Record Transaction Breakdown Information on page 865

Reference

Troubleshooting and Limitations on page 867

Concepts

Siebel Web Protocol Overview

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.

Siebel Web Recording Options and Run-Time Settings

Before recording a Siebel Web Vuser, we recommend that you 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.

Before replaying or load testing a Siebel Web Vuser, we recommend that you set the following Run-Time setting:

- ▶ In the Run-Time settings, clear the **Simulate a new user on each iteration** option in the Browser Emulation node.

Tasks

How to Record Transaction Breakdown Information

VuGen provides a diagnostic tool for understanding the transaction components in your test—**transaction breakdown**. Using transaction breakdown, you can determine where the bottlenecks are and the issues that need to be resolved.

When preparing your script for transaction breakdown, we recommend that you add think time steps at the end of each transaction using the ratio of one second per hour of testing. For more information about adding think time steps, see "How to Insert Steps into a Script" on page 155.

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=SWEEEntryForm", 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.

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations for Siebel Web Vuser scripts.

Back or Refresh Error

An error message relating to **Back or Refresh** typically has the following text:

We are unable to process your request. This is most likely because you used the browser `BACK` or `REFRESH` button to get to this point.

Cause: The possible causes of this problem may be:

- ▶ The SWEC was not correlated correctly for the current request.
- ▶ The SWETS was not correlated correctly for the current request.
- ▶ The request was submitted twice to the Siebel server without the SWEC being updated.
- ▶ A previous request should have opened a frame for the browser to download. This frame was not created on the server probably because the SWEMethod has changed since the recording.

Same Values

A typical Web page response to the **Same Values** error is:

```
@0'0'3'3''0`UC`1`Status`Error`SWEC`10`0`1`Errors`0`2`0`Level0`0`ErrMsg`The same values for 'Name' already exist. If you would like to enter a new record, please make sure that the field values are unique.`ErrCode`28591`
```

Cause: The possible cause of this problem may be that one of the values in the request (in the above example, the value of the Name field) duplicates a value in another row of the database table. This value needs to be replaced with a unique value to be used for each iteration per user. The recommended solution is to replace the row ID with its parameter instead insuring that it will be unique.

No Content HTTP Response

A typical HTTP response for a **No Content HTTP Response** type error is:

```
HTTP/1.1 204 No Content  
Server: Microsoft-IIS/5.0  
Date: Fri, 31 Jan 2003 21:52:30 GMT  
Content-Language: en  
Cache-Control: no-cache
```

Cause: The possible causes of this problem may be that the row ID is not correlated at all or that it is correlated incorrectly.

Restoring the Context

The typical Web page response to the **Restoring the Context** type error is:

```
@0'0'3'3''0`UC`1`Status`Error`SWEC`9`0`1`Errors`0`2`0`Level0`0`ErrMsg`An error happened during restoring the context for requested location`ErrCode`27631`
```

Cause: The possible causes of this problem may be that the rowid is not correlated or that it is correlated incorrectly.

Cannot Locate Record

The typical Web page response to the **Cannot locate record** type error is:

```
@0'0'3'3'0'UC'1'Status`Error`SWEC`23'0'2'Errors`0'2'0'Level0'0'ErrMsg`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.

32

SilverLight Protocol

This chapter includes:

Concepts

- ▶ Silverlight Protocol Overview on page 872

Tasks

- ▶ How to Import WSDL Files on page 873

Concepts

Silverlight Protocol Overview

Microsoft Silverlight is a web application framework that supports graphics, animations, and interactivity. VuGen's Silverlight protocol enables you to record applications built with Microsoft Silverlight. VuGen's Silverlight protocol includes the Web (HTTP/HTML) protocol as a subset as well as including a number of new functions, recording options, and run-time settings.

In order to record high level scripts, you can import WSDL files used by your application in the recording options.

Tasks

How to Import WSDL Files

The following steps describe how to import WSDL files into a Silverlight script manually or automatically. Alternatively, you can disable WSDL files and generate soap requests. All of these options are performed in the **Silverlight > Services** node of the **Recording Options Dialog Box**. For user interface details, see "Silverlight Services Node" on page 409.

- "Automatically locate WSDL files" on page 873
- "Manually locate WSDL files" on page 873
- "Disable WSDL files" on page 874
- "Advanced Security Settings" on page 874

Automatically locate WSDL files

To configure VuGen to automatically detect the WSDL files used by your script and attempt to locate them, select **Use WSDL files included in the script** and **Automatically detect WSDL files and import services during code generation**. If a WSDL is detected that cannot be imported, you will be notified in the Code Generation Notifications box.

Manually locate WSDL files

You can manually locate WSDL files in a number of ways from the Add Service Dialog Box. To locate a WSDL file whose URL is known, use the **URL** option. If the WSDL file is on your local machine, you the **File** option. To search for the WSDL in the WSDL History (a list of previously imported WSDLs), select **Previously Imported** and click ... to open the list.

For user interface details, see "Add / Edit Services Dialog Box" on page 410.

Disable WSDL files

You can disable WSDL files and generate SOAP requests instead. This results in a lower level script, however it does increase the performance of your script. To disable WSDL files, select **Do not use WSDL files**.

Advanced Security Settings

You can modify security and password settings in the Protocol and Security Scenario Data dialog box. For details, see "Protocol and Security Scenario Data Dialog Box" on page 412.

33

Tuxedo Protocols

This chapter includes:

Concepts

- ▶ Tuxedo Protocol - Overview on page 876
- ▶ Notes About Working with Tuxedo Scripts on page 877
- ▶ Defining Environment Settings for Tuxedo Vusers on page 878

Reference

- ▶ Tuxedo Buffer Data on page 880

Troubleshooting and Limitations on page 881

Concepts

Tuxedo Protocol - Overview

When you record a Tuxedo application, VuGen generates LRT functions that describe the recorded actions. These functions emulate communication between a Tuxedo client and a server. Each Tuxedo function begins with an **lrt**, **tp**, **tx**, or **F** prefix.

In the following example, VuGen recorded a client's actions in a Tuxedo bank application. The client performed an action of opening a bank account and specifying all the necessary details. The session was aborted when the client specified a zero opening balance.

```
lrt_abort_on_error();
lr_think_time(65);
tpresult_int = lrt_tpbegin(30, 0);
data_0 = lrt_tpalloc("FML", "", 512);
lrt_finitialize((FBFR*)data_0);

/* Fill the data buffer data_0 with new account information */
lrt_fadd_fid((FBFR*)data_0, "name=BRANCH_ID", "value=8",
LRT_END_OF_PARMS);
lrt_fadd_fid((FBFR*)data_0, "name=ACCT_TYPE", "value=C",
LRT_END_OF_PARMS);
lrt_fadd_fid((FBFR*)data_0, "name=MID_INIT", "value=Q", LRT_END_OF_PARMS);
lrt_fadd_fid((FBFR*)data_0, "name=PHONE", "value=123-456-7890",
LRT_END_OF_PARMS);

lrt_fadd_fid((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
New York, NY 10000", LRT_END_OF_PARMS);
lrt_fadd_fid((FBFR*)data_0, "name=SSN", "value=111111111", LRT_END_OF_PARMS);

lrt_fadd_fid((FBFR*)data_0, "name=LAST_NAME",
"value=Doe", LRT_END_OF_PARMS);

lrt_fadd_fid((FBFR*)data_0, "name=FIRST_NAME",
"value=BJ", LRT_END_OF_PARMS);

lrt_fadd_fid((FBFR*)data_0, "name=SAMOUNT",
"value=0.00", LRT_END_OF_PARMS);
```

```

/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, &data_0, &olen_2, 0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();

```

Notes About Working with Tuxedo Scripts

The following important notes should be reviewed before recording and running a tuxedo script:

- We recommend using the Tuxedo 6 protocol for recording Tuxedo 6.x and earlier, and the Tuxedo protocol for Tuxedo 7.x and higher.
- Before you record, verify that the Tuxedo 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.
- To run PeopleSoft-Tuxedo Vusers with Tuxedo 7.x, you must change the library extension in the *mdrv.dat* file as follows:

```

[PeopleSoft-Tuxedo]
WINNT_EXT_LIBS=lrt7.dll

```

Defining Environment Settings for Tuxedo Vusers

The following section describes the system variable settings for Tuxedo Vusers running on Windows and 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=bankfids,usysfids (PC)
setenv FLDTBLDIR $TUXDIR/udataobj:$TUXDIR/apps/bankapp (Unix)
setenv FIELDTBLS bank.fids,Usysfids (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=0x0002fffc7cb4e4a (PC)
setenv WSNADDR 0x0002fffc7cb4e4a (Unix)
setenv WSDEVICE /dev/tcp (Unix)
```

Reference

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.

```

/* Request CARRAY buffer 4 */
static const char sbuf_4[] =
"\xe2\x00\x00\x4\x3\x2\x1\x1\x0\x0\x0\xbc\x2\x0\x0\x0"
"T"
"\x0\x0\x4\x0"
"SCTX"
"\x4"
"CULT"
"\x2"
"PS"
"\x2"
"PS"
"\x1"
"7"
"\x0\x0\x0\x8"
"ALLPANIS"
"\x8"

```

Troubleshooting and Limitations

This section describes troubleshooting and limitations for Tuxedo and Tuxedo 6 Vusers.

- ▶ If you encounter problems recording or replaying Tuxedo applications, or the script is missing a call to `lrt_tpinitialize`, contact Customer Support to check which DLLs are used with the application.
- ▶ If the application uses `wtuxws32.dll`, instead of `libwsc.dll`, contact Customer Support to obtain a patch to enable the recording.
- ▶ If you encounter problems recording or running Tuxedo applications, check that the Tuxedo application runs without VuGen, and that the environment variables have been defined correctly. For more information, see "Tuxedo Buffer Data" on page 880. 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.

34

Web Protocols

This chapter includes:

Concepts

- ▶ Web Protocols Overview on page 884
- ▶ Web Vuser Technology on page 885
- ▶ Web Vuser Types on page 886
- ▶ Support for Push Technology on page 889
- ▶ Working with Cache Data on page 889
- ▶ Text and Image Verification on page 890
- ▶ Data Format Extensions on page 893
- ▶ Web Snapshots on page 894
- ▶ XML Pages on page 895

Tasks

- ▶ How to Add Text and Image Checks on page 896
- ▶ How to Convert Web Vuser Scripts into Java on page 897
- ▶ How to Insert Caching Functions on page 898

Reference

- ▶ Data Format Extension List on page 901

Concepts

Web Protocols Overview

You use VuGen to develop Web Vuser scripts. While you navigate through a site performing typical user activities, VuGen records your actions and generates a Vuser script. When you run the script, the resulting Vuser emulates a user accessing the Internet.

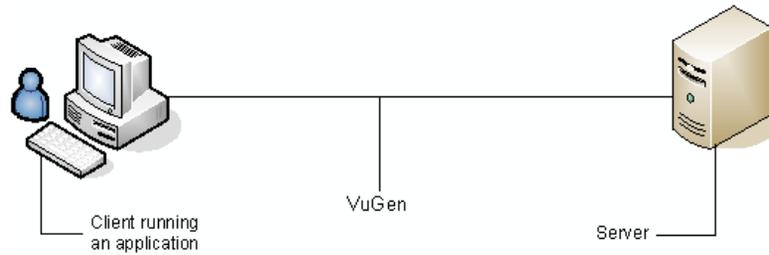
Suppose you have a Web site that displays product information for your company. The site is accessed by potential customers. You want to make sure that the response time for any customer query is less than a specified value (for example, 20 seconds)—even when a large number of users (for example, 200) 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

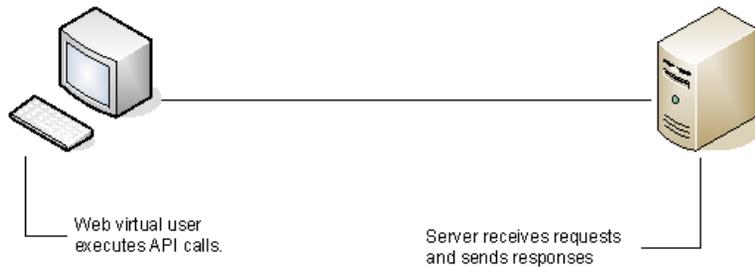
You can distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

Web Vuser Technology

VuGen creates Web Vuser scripts by recording the activity between a browser and a Web server. VuGen monitors the client (browser) end of the system and traces all the requests sent to, and received from, the server.



When you run a recorded Vuser script, the Vuser communicates directly with the server without relying on client software. Instead, the Vuser script executes calls directly to the Web server via API functions.



Web Vuser Types

When creating a new Web Vuser script, you can select one of the following types of Web Vusers:

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 let 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 "Recording Levels Overview" on page 312.

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.

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 **web_reg_cross_step_download** function in the *Online Function Reference* (**Help > Function Reference**).

Working with Cache Data

You can save stored data into your browser’s cache, and load it at a later point in the script.

To implement this within your script, you manually add the **web_dump_cache** and **web_load_cache** functions.

For task details, see "How to Insert Caching Functions" on page 898.

Dumping Information to the Cache

Transferring data to the cache is called dumping the information. You run the **web_dump_cache** function to create a cache file in the location specified in the **FileName** argument. You 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)
```

Text and Image Verification

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.

To add an image or text check, see "How to Add Text and Image Checks" on page 896.

Understanding Web Check Functions

When you add a text check, VuGen adds a **web_reg_find** function to your script. This function registers a search for a text string on an HTML page. Registration means that it does not execute the search immediately—it performs the check only after executing the next Action function, such as **web_url**. Note 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).

Data Format Extensions

VuGen supports the recording of many different types of data. New formats are constantly being created and VuGen must adapt to support them. Some formats are proprietary and use custom serialization, making it difficult for the user to understand the code due to binary and unformatted data. VuGen has developed a way to use Data Format Extensions (DFEs) to convert the code to more readable formats allowing you to parameterize and correlate this data.

Data Format Extensions work in ordered chains. VuGen attempts to deserialize the data using the DFEs in a chain one at a time. The extensions within a chain are each designed to deserialize/serialize a specific data format. If an extension does not convert the specified data, this data is passed to the next extension in the chain. If an extension does successfully convert the data, you can configure VuGen to either continue to run the rest of the DFE chain on the converted data, or to end the process.

There are several pre-defined chains which can be customized in the Data Format Extensions nodes of the Recording Options dialog box. For user interface details, see "Data Format Extensions - Chain Nodes" on page 344. There are a number of Data Format Extensions that come built-in to VuGen. For specific details about each DFE, see "Data Format Extension List" on page 901.

Additionally, advanced users can manually add and modify chains, as well as create new Data Format Extensions.

For more information, see the DFE SDK.

Web Snapshots

VuGen protocols based on the Web (HTTP/HTML) protocol have a unique snapshot pane. The snapshots display very detailed information about every web step. Each step can display snapshots taken during recording, replay or both. Each snapshot can be displayed using the HTML view or the more detailed HTTP view.

The HTTP view displays every HTTP transaction in the HTTP Flow or Tree (depending on the view). The transaction data is broken up into response and request data, headers, cookies, and query strings.

The screenshot shows the VuGen HTTP View interface. At the top, there are tabs for 'HTML View', 'HTTP View', 'Grid', and 'Tree'. The main area is titled 'HTTP Flow' and contains a table with the following data:

Path	Start Time	Response Time	IP	Port	Origin	Size [bytes]	Method	Type
/country/us/en/explore.htmr	4:37:33.375	203	15.201.57.2	80	Primary	1973	GET	text/html
/hho/hp_create	4:37:35.937	203	15.201.57.2	80	HTML	240	GET	text/html; ch
/hho/hp_create/	4:37:36.156	203	15.201.57.2	80	Redirection	49195	GET	text/html

Below the table, there are tabs for 'Raw Data', 'Response Body', 'Request Body', 'Headers', 'Cookies', and 'Query String'. The 'Request Body' tab is selected, showing the following request data:

```
Request
GET /country/us/en/explore.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Referer: http://www.hp.com/#Explore
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022;
InfoPath 2; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
Host: www.hp.com
Connection: Keep-Alive
Cookie: EMID= s_vi=[CS]v1|
25C14E2F85011AFc-40000110000056AF[CE]; lang=en-us;
cc=us; HP_EBUS=true; hp_cust_seg_sel=HHO;
hpcompc_usen=cartExists=false;
SMIDENTITY=iFlu3qg4ORE3XcMi8jflcFlaucy3lutwUnlc5pdzY
FtjHdqyAst1yIPlCoPgah4PrpPKdvk0FJ9lu9gPZOEIAoaDZv13LkI
NYAbxnaqfqsuJww/L0bqGXRVjs5v1R5+
isj8540vGV34hh3kjQk8GlaAS5L0y3ckvkzq6ESsGxGrPnp2Aqh3
MttHFqg1013afct8Uw0xlRdr53cl0fSiwFygQHlWwx1SCVDG2Y
FwjAU9mq54oA5HRmNld12QSPB3Z6cH1zBGOvMXLZ5HU2ad
UwnLHqM55figHlm9DoA1TKp8OYRZUBocqJ9JTzr8g4n50lvYdjF
6JHKdBcbjftPCLT0JYPKQWuPR0GRj5PptcMOWDgTA31XGk9sj
gUzj2qk2lQlllT02JYXodPB5TqpZ8MqCE8mybJ7MHlJEJ7eYmE
MFC1iNqL05mT34317V9jaQwdTUVFnX9F
...

```

The 'Response' pane shows the following response data:

```
Response
HTTP/1.1 200 OK
Date: Mon, 21 Jun 2010 11:36:21 GMT
Server: Apache
Accept-Ranges: bytes
Cache-Control: max-age=3600
Expires: Mon, 21 Jun 2010 12:36:21 GMT
Content-Encoding: gzip
Content-Length: 1973
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="en-us">
<head>
<title>HP Explore and Create</title>
</head>
<body class="blackBg">
<div class="centerContent">
<div class="mainContentArea">
<!-- BEGIN EXPLORE -->
<div id="mainPaddedPanel" style="position:
relative;">

```

Data can be displayed using a number of different editors: Text Editor, Hex Editor, and XML Editor.

Correlations and Parameters can be created on response data by selecting the desired text and right-clicking.

For data that is difficult to work with (such as binary data) VuGen offers a variety of Data Format Extensions that can transform certain data types into more readable formats. Data that has been formatted by a Data Format Extension can be displayed in its original or formatted state. For more information, see "Data Format Extensions" on page 893.

The new Web snapshot model is backward compatible with previous versions of LoadRunner, however some snapshot data may be missing. If this occurs, regenerate the script.

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.

Tasks

How to Add Text and Image Checks

There are a number of different types of checks that VuGen can add to your script. For background information, see "Text and Image Verification" on page 890.

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.



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 and complete the Find Text properties dialog box. For more details, hit F1 when in the dialog box to open the function reference.

Add other text checks (after recording)

- 1 Select the step that contains the image you want to verify.
- 2 Select **Insert > New Step**.
 - a For the `web_find` function, expand the **Web Checks** node and select **Text Check**.
 - b For the `web_global_verification` function, expand the **Services** node and select the function name.
- 3 Complete the dialog box. For more details, hit F1 when in the dialog box to open the function reference.

Add an image check (after recording)

- 1 Select the step that contains the image you want to verify.
- 2 Select **Insert > New Step > Web Checks > Image Check**.
- 3 Complete the Image Check Properties dialog box. For more details, hit F1 when in the dialog box to open the function reference.

How to Convert Web Vuser Scripts into Java

VuGen provides a utility that enables you to convert a script created for a Web Vuser into a script for Java Vusers. This also allows you to create a hybrid Vuser script for both Web and Java.

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.

How to Insert Caching Functions

This task describes how to use caching functions. Caching functions allow you to save stored data into your browser's cache, and load it at a later point in the script. For more information, see "Working with Cache Data" on page 889.

To use the caching functions:

- 1** Insert the **web_dump_cache** function into your script.
- 2** Run the script at least once.
- 3** Insert the **web_load_cache** function into your script before the Vuser actions.
- 4** Comment out the **web_dump_cache** function.
- 5** Run and save the script.

Example:

The following example illustrates a PeopleSoft Enterprise Vuser viewing the details of his paycheck.

```
Action()
{
//  web_add_cookie("storedCookieCheck=true; domain=pbntas05; path="");

web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",LAST);

    web_browser("signon.html",
        DESCRIPTION,
        ACTION,
        "Navigate=http://pbntas05:8200/ps/signon.html",
        LAST);
    lr_think_time(35);

    web_edit_field("userid",
        "Snapshot=t1.inf",
        DESCRIPTION,
        "Type=text",
        "Name=userid",
        ACTION,
        "SetValue={VuserName}",
        LAST);
```

```

web_edit_field("pwd",
    "Snapshot=t2.inf",
    DESCRIPTION,
    "Type=password",
    "Name=pwd",
    ACTION,
    "SetValue=HCRUSA_KU0007",
    LAST);

lr_start_transaction("login");
    web_button("Sign In",
        "Snapshot=t3.inf",
        DESCRIPTION,
        "Type=submit",
        "Tag=INPUT",
        "Value=Sign In",
        LAST);
lr_end_transaction("login", LR_AUTO);

web_image_link("CO_EMPLOYEE_SELF_SERVICE",
    "Snapshot=t4.inf",
    DESCRIPTION,
    "Alt=",
    "Name=CO_EMPLOYEE_SELF_SERVICE",
    "Ordinal=1",
    ACTION,
    "ClickCoordinate=10,10",
    LAST); ...

web_text_link("Sign out",
    "Snapshot=t7.inf",
    DESCRIPTION,
    "Text=Sign out",
    "FrameName=UniversalHeader",
    ACTION,
    "UserAction=Click",
    LAST);

/*web_dump_cache("paycheck","FileName=c:\\{VuserName}paycheck",
"Replace=yes", LAST);*/
    return 0;
}

```

Reference

Data Format Extension List

The following table lists the Data Format Extensions that come built-in to VuGen. For more information about Data Format Extensions, see "Data Format Extensions" on page 893.

Data Format Extension	Description
Base64 Extension	Decodes strings that are encoded with a BASE64 encoder.
URL Encoding Extension	Decodes strings that are encoded with URL encoding format.
JSON Extension	Transforms JSON data to XML format.
XML Validator Extension	Receives data and checks to see if it conforms with XML syntax. This check allows VuGen to perform correlations based on xpath and to display snapshot data in an xml viewer.

35

Web Services - Adding Script Content

This chapter includes:

Concepts

- ▶ Web Service Testing Overview on page 910
- ▶ Adding Web Service Script Content Overview on page 910
- ▶ Special Argument Types on page 918
- ▶ Generating Service Requirements and Tests Overview on page 922
- ▶ Server Traffic Scripts Overview on page 923

Tasks

- ▶ How to Add Content on page 928
- ▶ How to Assign Values to XML Elements on page 931
- ▶ How to Generate a Test Automatically on page 932
- ▶ How to Create a Script by Analyzing Traffic on page 934
- ▶ How to Create Business Components in VuGen on page 937
- ▶ How to Create Business Components in Application Lifecycle Management on page 939

Reference

- ▶ Add Script Content User Interface on page 941
- ▶ Aspect Reference on page 959
- ▶ Test Generator Wizard User Interface on page 961
- ▶ Analyze Traffic User Interface on page 966

Concepts

Web Service Testing Overview

SOA systems are based on Web Services, self-contained applications that can run across the Internet on a variety of platforms. The services are built using Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP). They serve as building blocks 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.

Adding Web Service Script Content Overview

Web Services scripts let you test your environment by emulating Web Service clients.

After creating an empty Web Services script, as described in "New Virtual User Dialog Box" on page 117, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

A Service Test license provides an additional method for creating scripts—the SOA Test Generator.

This section also includes:

- ▶ "Recording a Web Services Script" on page 911
- ▶ "Adding New Web Service Calls" on page 911
- ▶ "Importing SOAP Requests" on page 912
- ▶ "Analyzing Server Traffic" on page 913
- ▶ "Business Process Testing" on page 913

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.

When you record an application, you can record it with or without a Web Service WSDL file. If you include a WSDL file, VuGen allows you to create a script by selecting the desired methods and entering values for their arguments. VuGen creates a descriptive script that can easily be updated when there are changes in the WSDL.

If you record a script without previously importing a service (not recommended) VuGen creates SOAP requests instead of Web Service call steps. SOAP request arguments are less intuitive and harder to maintain.

For more information, see "Record a session - optional" on page 928.

Adding New Web Service Calls

You can create a script is by manually adding Web Service calls. You design the call based on an operation, transport, arguments, and other properties.

For more information, see "Add a new service call- optional" on page 929.

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. You can view the values in the **Step Properties** tab under the Input Arguments node.

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**).

For more information, see "Import SOAP - optional" on page 930.

Analyzing Server Traffic

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

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

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server.

For more information, see "How to Create a Script by Analyzing Traffic" on page 934.

Business Process Testing

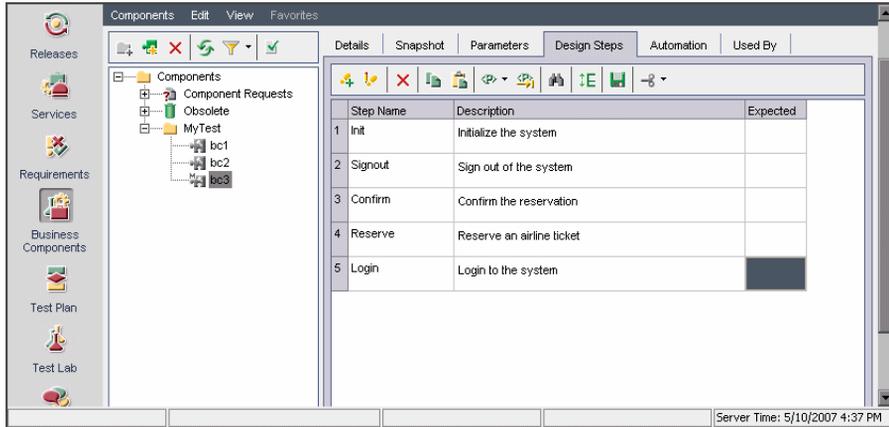
BPT (Business Process Testing) is a methodology in which several tests are combined to create a complete business process. The BPT user composes a complete test by combining a series of test components with data flow between them.

Note: The BPT features are only available with a Service Test license. For details, contact HP Support.

Components are comprised of steps. For example, a login component's first step may be to open the application. Its second step could be entering a user name. Its third step could be entering a password, and its fourth step could be clicking the **Enter** button.

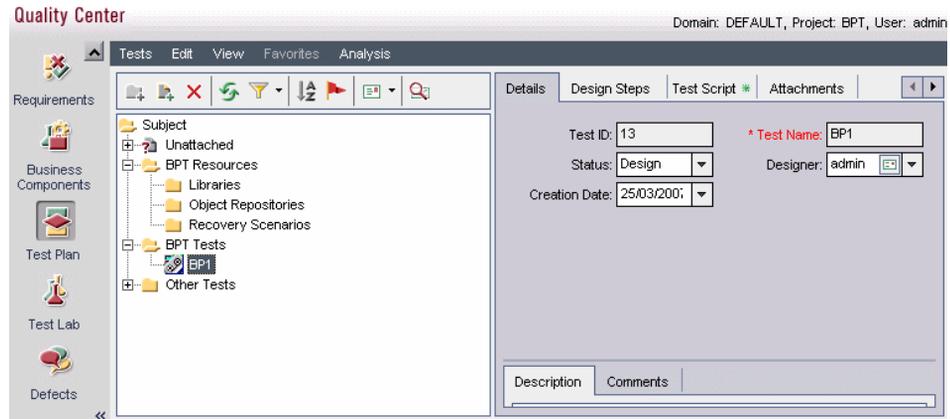
You can create business process test components from within ALM or through VuGen with Service Test.

In Application Lifecycle Management, non-technical SMEs (Subject Matter Experts) define design steps that are required for the test.



After these are defined, the technical engineer creates a script through Service Test that performs the desired actions. For more information about creating Business Components from within Application Lifecycle Management, see the *Business Process Testing user guide*.

You create components in Service Test by recording a session with your application or manually editing a script. You can add checkpoints, parameterize selected items, and enhance the component with flow statements and other testing functions. You then save the component to a project in Application Lifecycle Management. A Subject Matter Expert using Business Process Testing in Application Lifecycle Management combines your saved components into one or more business process tests, which are used to check that the application behaves as expected.



When you create parameters for your business component in Application Lifecycle Management, they will be available in Service Test's parameter list.

Some of the advantages of working with a BPT module over individual scripts are:

- Enables less-technical subject matter experts to create tests
- Enables structured automated testing
- Reduces the duplication of effort when combining manual tests with automatic scripts
- Allows component reusability to speed-up the automation process
- Provides the ability to pass parameters from one step to another within your business process. You can save the output of a step to a parameter and use it as an input value for subsequent steps.
- Simplifies on-going test maintenance

- ▶ Minimizes time-to-test

For task details, see Chapter 35, "How to Create Business Components in VuGen."

For more information about creating BPT components in Application Lifecycle Management, see the *Business Process Testing User Guide*.

Script Integration

You can use the completed script to test your system in several ways:

- ▶ **Functional Testing.** Run the script to see if your Web services are functional. You can also check to see if the Web service generated the expected values. For more information, see Chapter 37, "Web Services - Preparing Scripts 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 configuration. For more information, see the *HP Business Service Management* documentation.

An add-on for HP Application Lifecycle Management, Service Test Management, lets you manage SOA testing by allowing you to import, store, and define services in Application Lifecycle Management. For more information, contact your HP representative.

Web Service Call Attachments

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

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

The formats used for passing a SOAP message together with binary data are MIME (Multipurpose Internet Mail Extensions) and the newer, more efficient DIME (Direct Internet Message Encapsulation) specifications. VuGen supports DIME for all toolkits, but MIME only for the Axis toolkit. To use MIME attachments for the .NET toolkit, see "Including MIME Attachments" on page 1048.

VuGen supports the sending and receiving of attachments with SOAP messages. You can send Input (Request) or save Output (Response) attachments. For task details, see "Add attachments- optional" on page 930.

Output attachments are used to save the response as an attachment. You can choose one of the following options: **Save All Attachments** or **Save Attachment by Index**.

When you specify **Save All Attachments**, VuGen creates three parameters for each attachment based on the parameter name that you specify: a parameter containing the attachment data, the content type of the attachment, and a unique ID for the attachment.

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

```
MyParam_1
MyParam_1_ContentType
MyParam_1_ContentID
```

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

Special Argument Types

VuGen handles special argument types such as derived, recursive, choice, and optional elements.

This section includes:

- "Derived Types" on page 918
- "Abstract Types" on page 918
- "Optional Elements" on page 919
- "Choice Optional Elements" on page 920
- "Recursive Elements" on page 920

Derived Types

VuGen supports WSDLs with derived types. When setting the properties for a Web Service Call, VuGen allows you to use the base type or derived type for the argument. After you select a type, VuGen updates the argument tree node to reflect the new type. For details, see "<Input Argument Name> Node" on page 949.

Abstract Types

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

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

Optional Elements

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

```
minOccurs='0'
nillable='true'
```

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

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

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

The following table indicates the availability of the options:

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

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

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

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

Choice Optional Elements

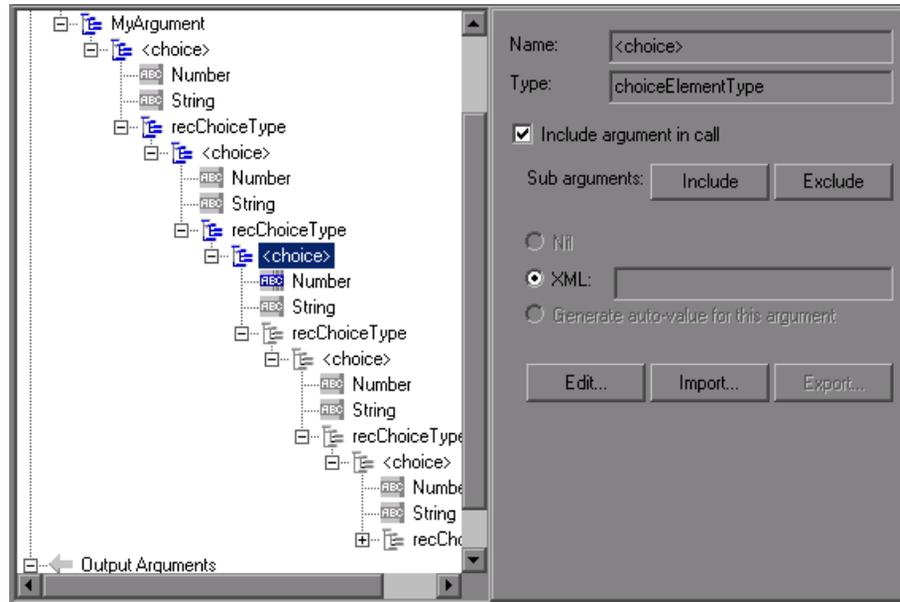
A Choice element in a WSDL defines a set of elements where only one of them appears in the SOAP message. In some cases, one of the Choice elements is optional, while the others are not. You can select the Choice element and still prevent its optional element from appearing in the SOAP envelope. In Tree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

Recursive Elements

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

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

In the following example, three levels of the Choice argument are included—the rest are not. Excluded nodes are grayed out.



Base 64 Arguments

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

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

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

For details, see "Process Base64 Data - Simple Data Dialog Box" on page 957.

Generating Service Requirements and Tests Overview

To test your SOA environment, you can create tests manually, or use the SOA Test Generator to automatically generate scripts.

Note: The Automatic Script Generator is only available with a Service Test license. For details, contact HP Support.

This section describes how to use the SOA Test Generator. For information on creating tests manually, see "How to Add Content" on page 928.

The SOA Test Generator guides you through the process of creating scripts to test your services. Through the wizard, you indicate which aspects of the service you want to test. These aspects include interoperability with different toolkits, boundary testing, and standard compliance.

After you select the testing aspects, VuGen with Service Test automatically generates one or more scripts for each of the aspects.

Testing Aspects

The SOA Test Generation wizard helps you create requirements and tests that verify different aspects of your service. Service Test Management creates a separate requirement for each aspect and sub-aspect, and a separate test for each sub-aspect. It only creates a separate test for an aspect if it has no sub-aspects.

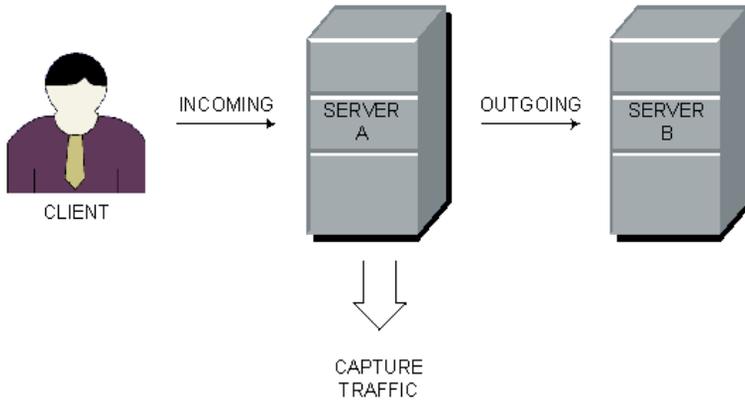
By default, Service Test Management supports the following testing aspects. For more information, see the "Aspect Reference" on page 959.

Server Traffic Scripts Overview

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

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

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server. The steps in creating a script by analyzing server traffic are described below in "How to Create a Script by Analyzing Traffic" on page 934.



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**.

Capture Files

A capture file is a trace file containing a log of all TCP traffic over the network. Using a sniffer application, you obtain a dump of all of the network traffic. The sniffer captures all of the events on the network and saves them to a capture file.

To generate a smaller, more manageable script, try to capture the network traffic only for the time that you perform actions in your application.

Note: Capture files do not contain loopback network traffic.

You can obtain a capture file using the command line utility or any existing capture tool.

The VuGen command line utility, **lrtcpdump**, is located in the product's **bin** directory. There is a separate utility for each of the platforms: **lrtcpdump.exe** (Windows), **lrtcpdump.hp9**, **lrtcpdump.ibm**, **lrtcpdump.linux**, and **lrtcpdump.solv4**.

External Capture Tools

Most UNIX operating systems have a built-in version of a capture tool. In addition, there are many downloadable capture tools such as **Ethereal/tcpdump**.

When using external tools, make sure that all packet data is being captured and none of it is being truncated.

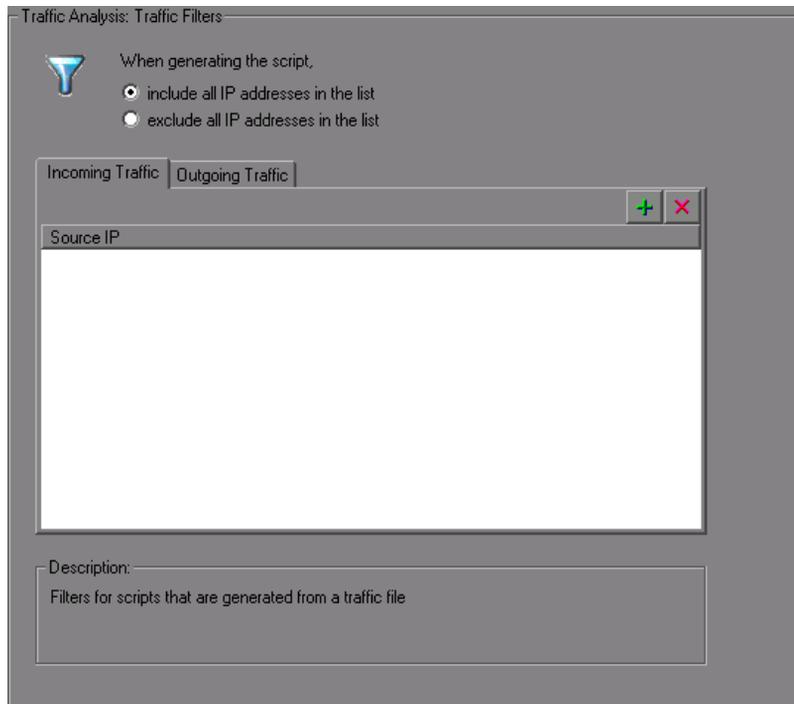
Certain capture utilities require additional arguments. For example, **tcpdump** requires the **-s 0** argument in order to capture the packets without truncating their data.

Filtering Traffic

You can provide a filter to drill down on specific requests going to or from a server, by specifying its IP address and port.

Tip: Several external capture tools allow you to filter the IP addresses while capturing the traffic.

You filter the requests by choosing the relevant host IP addresses. The filter can be inclusive or exclusive—you can include only those IPs in the list, or include everything except for those IPs that appear in the list.



For more information, see "Filter the IP addresses - optional" on page 936.

Data on Secure Servers

To analyze traffic from a secure server, you must provide a certificate containing the private key of the server.

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

For more information, see "Configure the SSL - optional" on page 937.

Tasks

How to Add Content

This task describes how to add content, such as Web Service calls, to a script.

This task includes the following steps:

- ▶ "Prerequisites" on page 928
- ▶ "Record a session - optional" on page 928
- ▶ "Add a new service call- optional" on page 929
- ▶ "Import SOAP - optional" on page 930
- ▶ "Analyze server traffic - optional" on page 931

Prerequisites

- 1** Create an empty Web Services script. Click **File > New** and choose the **Web Services** Vuser type. You can create either a single or multi-protocol type script.

Record a session - optional

- 1** Click the **Start Record** button or Ctrl+R to open the Specify Services screen.
For task details, see "Select Services Screen" on page 941.
- 2** Add services to the list. Click **Import** to load a WSDL for the test. Indicate the location of the WSDL file.
For user interface details, see "Import Service Dialog Box" on page 1008.
- 3** Click **Next**. Specify the location of the application and any other relevant arguments. See the "Start Recording Dialog Box" on page 942.
For user interface details, see "Start Recording Dialog Box" on page 942.

Add a new service call- optional

- 1** Import a service. Click **Manage Services** to access the Import dialog box.
For user interface details, see "Import Service Dialog Box" on page 1008.
- 2** Click the cursor at the desired location in your script (Script view) or in the test steps (Tree view).
- 3** Click the **Add Service Call** button. The New Web Service Call dialog box opens.
- 4** In the Select Web Service Call section, select a **Service**, **Port Name**, and **Operation**.
- 5** To specify an endpoint other than the default **Target Address**, select **Override Address** and insert the new endpoint to which you want to submit the requests.
- 6** Expand the nodes and specify argument values. To create sample values for all Input arguments, select the **Input Arguments** node and click **Generate**. To edit, import, or export the element's XML structure, see "How to Assign Values to XML Elements" on page 931.
- 7** To parameterize an input argument, click the node and select the **Value** option. Click the ABC icon and proceed with parameterization. For more information, see "Parameters" on page 257.
- 8** Select the **Transport Layer Configuration** node to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see "How to Send Messages over JMS" on page 1055.

Add attachments- optional

- 1 To add an attachment to an input argument, choose an operation in the left pane. Select **Add to request (Input)**. VuGen prompts you to enter information about the attachment and adds it to the method's tree structure. For details, see the "Add Input Attachment Dialog Box" on page 954.
- 2 To specify an output attachment through which to store output arguments, choose an operation in the left pane. Select **Save received (Output)**. Select the desired option: **Save All Attachments** or **Save Attachment by Index** based on their index number—beginning with 1. For details, see "Web Service Call Attachments" on page 916.
- 3 To edit the properties of either an Input or Output attachment, click the attachment in the left pane, and enter the required information in the right pane.

Specify SOAP headers- optional

Select the **Custom SOAP Header** node in the left pane and enable the **Use SOAP header** option. You must individually specify SOAP headers for each element. To compose your own, click **Edit** and edit the XML. To import an XML file for the SOAP header, click **Import**.

Import SOAP - optional

- 1 Import a service if one is available. Click **Manage Services** to access the Import dialog box. For more information, see the "Import Service Dialog Box" on page 1008.
- 2 Click the **Import SOAP** button to open the Import SOAP dialog box.
- 3 Browse for the XML file that represents your SOAP request.
- 4 Select the type of step you would like to generate: **Create Web Service Call** or **Create SOAP Request**. In order to create a Web Service Call, you must first import at least one WSDL that describes the operation in the SOAP request file. To view the SOAP before loading it, click **View SOAP**.
- 5 Click **Load** to import the XML element values.

For a **Web Service Call**, set the properties for the Service call as described in the "New Web Service Call Dialog Box" on page 945.

For a **SOAP Request**, provide the URL and the other relevant parameters.

- 6 For a Web Service Call, if there are multiple services with same operation (method) names, select the service whose SOAP traffic you want to import.
- 7 Click **OK** to generate the new step within your script.

Analyze server traffic - optional

To create a script by analyzing a file containing a dump of the server traffic, click **Analyze Traffic**.

For details, see ("Server Traffic Scripts Overview" on page 923.

How to Assign Values to XML Elements

This task describes how to work with XML elements by manually editing the code, importing an external file, and exporting the XML for later use.

This task includes the following steps:

- "Prerequisites" on page 932
- "Select the element" on page 932
- "Import a file - optional" on page 932
- "Edit the XML elements - optional" on page 932
- "Export a file - optional" on page 932

1 Prerequisites

Import a service and create a new Web Service call. Alternatively, select a step and click the **Step Properties** tab.

2 Select the element

In the left pane, select a complex type or array argument. In the right pane, click **XML**. The XML field shows the XML code as a single string.

3 Import a file - optional

To import a previously saved XML file, click **Import** and specify the file's location.

4 Edit the XML elements - optional

To edit the XML structure and element values, click **Edit**. The XML Editor opens. To import a previously saved XML file, click **Import File**.

- ▶ To manually edit the code, click the **Text View** tab.
- ▶ To modify the XML through a graphical interface, click on **Tree View**. Use the shortcut menu to add children and sibling elements and rename the node. Click **Insert** from the shortcut menu to add a new element before or after the selected one.

5 Export a file - optional

To save your XML data to a file so it can be used for other tests, click **Export** and specify a location.

How to Generate a Test Automatically

This task describes how to create requirements or tests for checking your service.

This task includes the following steps:

- ▶ "Record a session - optional" on page 928
- ▶ "Import a Service - optional" on page 936

- "Specify traffic information" on page 936
- "Specify a location" on page 934
- "Complete the test generation" on page 934
- "Open the scripts" on page 934

1 Open the wizard

Select **File > New** to open the New Virtual User dialog box. Select **SOA Test Generator** in the left pane and click **Create**.

2 Add a service

Proceed to the next screen and click **Add** to import at least one service. If your service is not ready yet, you can use an emulated service. For details, see "How to Add and Manage Services" on page 991. Click **Next**.

3 Select testing aspects

Expand the nodes and select the desired testing aspects. For aspect details, see "Aspect Reference" on page 959. Click **Next**.

4 Specify a location

Specify a test name and a location for the test scripts: **HP ALM** or a **local file system**. If you specified ALM, click **Connect** to log on to the server and **Browse** to locate the test node.

5 Complete the test generation

Review the summary and include or exclude any scripts from the generation. Click **Generate**.

6 Open the scripts

In the final screen, review the list of generated scripts and indicate which ones to open. Click **Finish**.

How to Create a Script by Analyzing Traffic

This task describes how to create a script using a network traffic file.

This task includes the following steps:

- "Prerequisites" on page 928
- "Record a session - optional" on page 928

- "Import a Service - optional" on page 936
- "Specify traffic information" on page 936
- "Filter the IP addresses - optional" on page 936
- "Configure the SSL - optional" on page 937

1 Create a capture file on a Windows Platform - optional

Create a capture file containing a log of all TCP traffic over the network on a Windows platform. Use a downloadable capture tool or use the tool provided in the product's **bin** folder, **lrtcpdump.<platform>**.

- a** Run the capture utility in a command window **lrtcpdump -f<file_name>.cap**. **lrtcpdump** prompts you to select a network card.
- b** Type in the number of the interface card (if there are multiple ones.) and click **Enter**.
- c** Perform typical actions within your application.
- d** Return to the command window and click **Enter** to end the capture session.
- e** Place the capture file on the network in a location accessible to the machine running VuGen.

2 Create a capture file on a UNIX Platform - optional

Create a capture file containing all TCP traffic over the network on a UNIX platform.

- a** Locate the appropriate **lrtcpdump** utility for your platform in the product's **bin** 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.
- b** Switch to the root user to provide execution permissions: **chmod 755 lrtcpdump.<platform>**
- c** If there are multiple interface cards, **lrtcpdump** uses the first one in alphabetical order. To get a complete list of the interfaces, use the **ifconfig** command.

- d** Run the utility with its complete syntax, specifying the interface and file name. For example, `lrtcpdump.hp9 -ietho -f<file_name>.cap`. The capturing of the network traffic begins.
- e** Perform typical actions within your application.
- f** Return to the window running `lrtcpdump` and follow the instructions on the screen to end the capture session.
- g** Place the capture file on the network in a location accessible to the machine running VuGen.

3 Open the Analyzing Traffic wizard

Click the **Analyze Traffic** button or select **Vuser > Analyze Traffic**. For details, see "Select Services Screen" on page 941.

4 Import a Service - optional

Add one or more services to the list (optional). Click **Import** to load a WSDL file. For details, see the "Import Service Dialog Box" on page 1008.

Click **Next**.

5 Specify traffic information

Specify a capture file and the section of the script into which you want to load the traffic: **vuser_init**, **Action**, or **vuser_end**.

Indicate whether you want to analyze **Incoming** or **Outgoing** traffic. Specify the server whose traffic you want to analyze.

For details, see "Start Recording Dialog Box" on page 942.

6 Filter the IP addresses - optional

Click the **Filter Options** button to open the Recording options and indicate which IP addresses to ignore or include.

For details, see the Recording Options.

7 Configure the SSL - optional

Click the **SSL Configuration** button to add SSL certificates. This is necessary in order to analyze traffic from a secure server.

For details, see the "SSL Configuration Dialog Box" on page 969.

How to Create Business Components in VuGen

This task describes how to create business components in VuGen. This feature is only available with a Service Test license. For details, contact HP Support.

This task includes the following steps:

- "Prerequisites" on page 938
- "Create a new business component" on page 938
- "Add content" on page 938
- "Save the test" on page 938

1 Prerequisites

Make sure you have a connection to Application Lifecycle Management. For details, see "Managing Scripts Using ALM Overview" on page 246.

2 Create a new business component

Select **File > Business Component > New**. The New Business Component dialog box opens. Create a new Web Services script.

3 Add content

Add script content by recording, importing SOAP, or by manually adding Web Service calls. For details, see "Adding Web Service Script Content Overview" on page 910.

4 Parameterize the arguments - optional

Parameterize the desired arguments. If you want to share the parameter between business components in Application Lifecycle Management, create a BPT type parameter. For details, see "Parameter Types" on page 260.

5 Save the test

Save the script to the desired location in Application Lifecycle Management. When you create a business component in Service Test, it is regarded as an automated test in Application Lifecycle Management.

To convert an existing Vuser script to a Business Component, select **File > Business Component > Save as Business Component** and save the script to the desired location in Application Lifecycle Management.

6 Edit the business component - optional

Select **File > Business Component > Open**. Specify a script to open. Edit the script, set parameters, and add content as you would with any other script. Save the script.

How to Create Business Components in Application Lifecycle Management

The following section describes briefly how to create a business component in Application Lifecycle Management and automate it to be compatible with VuGen. For complete information about working with Business Components in Application Lifecycle Management, see the *Business Process Testing User Guide*.

This feature is only available with a Service Test license. For details, contact HP Support.

This task includes the following steps:

- "Create a new business component" on page 940
- "Create steps" on page 940
- "Automate the business component" on page 940
- "Edit the business component - optional" on page 940

1 Create a new business component

Select the **Business Components** module in Application Lifecycle Management and select **Component > New Component**.

2 Create steps

Select the **Design Steps** tab, create new manual steps, and define parameters (optional).

3 Automate the business component

In the **Design Steps** tab, select **Automate Component** (rightmost button) > **Service Test**.

4 Edit the business component - optional

Select the business component in the tree hierarchy. Select the **Automation** tab and click **Launch**. The script opens in VuGen.

Edit the script, set parameters and add content as you would with any other script.

For details about working with Business Components in Application Lifecycle Management, see the *Business Process Testing User Guide*.

Reference

Add Script Content User Interface

This section includes (in alphabetical order):

- Select Services Screen on page 941
- Start Recording Dialog Box on page 942
- Import SOAP Dialog Box on page 944
- New Web Service Call Dialog Box on page 945
- Add Input Attachment Dialog Box on page 954
- Add Array Elements Dialog Box on page 956
- Process Base64 Data - Simple Data Dialog Box on page 957
- Process Base64 Data - Complex Data Dialog Box on page 958
- Generate Test Generation Wizard on page 962
- Specify Services Screen on page 967
- Specify Traffic Information Screen on page 968
- SSL Configuration Dialog Box on page 969

Select Services Screen

This dialog box enables specify the basic details needed to begin recording a script.

To access	Start Record button
Relevant tasks	"Record a session - optional" on page 928

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

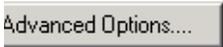
UI Elements (A-Z)	Description
Details...	Opens the Manage Services dialog box for providing further information about the service. For more information, see "Manage Services Dialog Box" on page 1003.
Import	Opens the Import Service dialog box. For more information, see the "Import Service Dialog Box" on page 1008. For user interface details, see the "Import Service Dialog Box" on page 1008.
Delete	Removes the selected service from the list.
<services list>	A list of the available services: <ul style="list-style-type: none"> ▶ Service Name. The native name of the service. ▶ WSDL Location. The source of the WSDL.

Start Recording Dialog Box

This dialog box enables specify the basic details needed to begin recording a Web Services script.

To access	Start Record button, Next
Relevant tasks	"Record a session - optional" on page 928

User interface elements are described below:

UI Elements (A-Z)	Description
	<p>Opens the Recording Options dialog box. For user interface details, see "Recording Options" on page 307.</p>
<p>Record default Web browser</p>	<p>Records the default browser actions. Specify the starting URL or click the Browse button to navigate to a location.</p>
<p>Record any application</p>	<p>Records any Win32 application. You can also specify the following:</p> <ul style="list-style-type: none"> ▶ Program to record. Select the browser, internet application, or Win32 application to record ▶ Program arguments (Win32 Applications only). Command line arguments for the application. For example, if you specify plus32.exe with the command line options peter@neptune, it connects the user Peter to the server Neptune when starting plus32.exe. ▶ Working directory. A working directory for the application (only when required by the application).
<p>Record into action</p>	<p>The section into which you want to record: vuser_init, Action, or vuser_end. For actions you want to repeat, use the Action section. For initialization steps, use vuser_init.</p>
<p>Record application startup</p>	<p>In the following instances, it may not be advisable to record the startup:</p> <ul style="list-style-type: none"> ▶ 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.

Import SOAP Dialog Box

This dialog box enables you to create a test step based on a SOAP file.

To access	Use one of the following:" <ul style="list-style-type: none"> ➤ Click  Import SOAP ➤ SOA Tools > Import SOAP
Relevant tasks	"Import SOAP - optional" on page 930 "Importing SOAP Requests" on page 912

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

UI Elements (A-Z)	Description
	Browse. Locate the XML file containing SOAP traffic.
	Loads the element values from the SOAP file.
	Opens the Manage Services dialog box for importing and configuring services.
	Opens the SOAP file in a browser for viewing.
<Call type>	The type of call to generate in the script/test: <ul style="list-style-type: none"> ➤ Web Service Call. Requires the import of a service. ➤ SOAP Request. Generates a soap_request step in the script.
SOAP Request Properties	The properties of the SOAP request (only visible for SOAP Request type calls). Specify the following: <ul style="list-style-type: none"> ➤ URL. The URL or IP address of the server to which to submit the request. ➤ SOAP Action. The SOAP action to include in the request (applicable if there are multiple actions). ➤ Response Parameter. A parameter name to store the response of the SOAP or Web Service call request.

New Web Service Call Dialog Box

This dialog box lets you create and configure a new Web Service call.

To access	Click  Add Service Call
Important Information	To access the Web Service call properties for existing Web Service calls, select a step in Tree view and choose Properties from the shortcut menu. For background information, see
Relevant tasks	"Add a new service call- optional" on page 929

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

UI Elements (A-Z)	Description
<service argument tree> (left pane)	An expandable tree hierarchy of the Service containing the following nodes: <ul style="list-style-type: none"> ▶ <operation name> ▶ Transport Layer Configuration ▶ Custom SOAP Header ▶ Input Arguments ▶ Output Arguments
<parameter values> (right pane)	Enables you to set and select values for each of the left pane's nodes.
Select Web Service Call	Lets you set the following items: <ul style="list-style-type: none"> ▶ Service. A dropdown list of all of the imported services, with the name derived from the WSDL. ▶ Port Name. A dropdown list of available ports through which to send the request. ▶ Operation. A dropdown list of the service's operations. ▶ Target Address. The default endpoint of the service. ▶ Override Address. Allows you to enter an alternate endpoint address in the Target Address box.

<Operation Name> Node

Allows you to generate sample values for the operation's input arguments and add attachments.

User interface elements are described below:

UI Elements	Description
Method	The name of the selected operation (read-only).
Generate auto-values for input arguments	Automatically creates sample values for all of the input arguments, based on their data type.
Attachments	Handles input and output attachments: <ul style="list-style-type: none"> ➤ Add to Request (Input). Attaches a file or parameter value to the request. ➤ Save Received (Output). Saves the response to a parameter.
Step properties	Lists the following service call properties and their values: <ul style="list-style-type: none"> ➤ WSDL file location ➤ Service name ➤ Port name ➤ Target address ➤ SOAP action ➤ SOAP namespace

Transport Layer Configuration Node

User interface elements are described below:

UI Elements	Description
HTTP/S Transport	Sets the transport method to HTTP or HTTPS transport.
Async Support	<p>Marks the Web Service call as an asynchronous message activated by an event:</p> <p>Async Event. An arbitrary name for the event.</p> <p>Note: Add a Web Service Wait For Event step to the script, to instruct the replay engine to wait for the event.</p>
WSA Support	<p>Enables WS-Addressing. Use one of the following options for a reply:</p> <ul style="list-style-type: none"> ➤ WS-A Reply. An IP address of the server to reply to when the event occurs. ➤ Autodetect. Reply to the current host when the event occurs. This is useful when running the same script on several machines. <p>Tip: To use WS-Addressing calls in synchronous mode, leave the Async Event box empty. In Script view, remove the AsyncEvent argument. This instructs the replay to block script execution until the complete response is received from the server.</p>
JMS Transport	<p>Sets the transport method to JMS for <i>synchronous</i> messages. For details, see "Testing Web Service Transport Layers Overview" on page 1025.</p> <p>Note: For JMS <i>asynchronous</i> messages, manually add a JMS Send Message Queue or JMS Receive Message Queue step to the script, to set up the message queue information.</p>
Override JMS Queues	Enables you to provide the request and response queues.
Request Queue	The queue name for the request message.
Response Queue	The queue name for the response message.

Custom SOAP Header Node

Lets you specify additional application-generated header elements to include in the SOAP envelope of an HTTP message. For task details see "Specify SOAP headers- optional" on page 930.

User interface elements are described below:

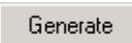
UI Elements	Description
	Opens an XML editor that lets you view and edit the SOAP header XML code.
	Opens the Select XML File to Import dialog box.
	Opens the Export SOAP Header into File dialog box.
	Opens the Select or Create New Parameter dialog box.
Use SOAP Header	Includes a SOAP header in the HTTP request.
Header	The header source: <ul style="list-style-type: none"> ▶ For an imported file: Header element as it appears in the imported file. ▶ For a parameter: The parameter name (in curly brackets)

Input Arguments Node

Lets you set the properties and generate values for all input arguments.

User interface elements are described below:

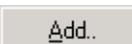
UI Elements	Description
	Includes all of the method's arguments in the Web Service call.
	Resets the arguments to their original state. It removes their inclusion in the call, and sets them to the values in the WSDL.

UI Elements	Description
	Generates sample data for all of the input arguments.
	Opens the pane for editing the selected argument's value.
Name	The name of the operation (read-only)
Argument List	A list of the input arguments. <ul style="list-style-type: none"> ➤  Simple parameters ➤  Arrays (shows the top level only).

<Input Argument Name> Node

When selecting an input argument, the right pane allows you to specify argument values.

User interface elements are described below:

UI Elements	Description
	Opens the Add Array Elements Dialog Box for adding a new array element to the input argument (only visible when selecting a parent node in the input array). For details, see the "Add Array Elements Dialog Box" on page 956
	Removes the selected array element in the input argument (only visible when selecting a parent node in the input array).
	Includes the sub-arguments of the selected argument, in the Web Service call. This is only enabled for an argument with sub-arguments with the Include argument in call option enabled.
	Excludes the sub-arguments of the parent argument, from the Web Service call.
	Opens an XML editor for editing the XML code containing the argument values. The only changes saved are the element values and the number of array elements.

UI Elements	Description
	Opens the Select XML File to Import dialog box,
	Opens the Export argument XML into file dialog box.
	Opens the Select or Create Parameter dialog box.
Name	The name of the argument or array.
Include argument in call	Include the argument in the call. For arrays, click Include to add the sub arguments to the call. To exclude all omissible arguments, click Exclude .
Type	The argument type as defined in the WSDL. When the WSDL contains derived types, this box becomes a drop down list. For details, see "Derived Types" on page 918.
Nil	Sets the Nillable attribute to true .
XML (for arrays only)	<ul style="list-style-type: none"> ▶ XML. Enables the Edit, Import, and Export buttons. By editing the XML, you can manually insert argument values. Click on the ABC icon to replace the entire XML structure with a single XML type parameter. Note: This import operation handles XML files that were previously exported—not standard SOAP files. ▶ Generate auto-value for this argument. Inserts automatic values for all children elements. ▶ Add/Delete. Adds or removes elements from the array.
Value (for non -array elements)	The argument value. To parameterize this value, click the ABC icon (only available for non-arrays).
Generate auto-value for this argument	Generates a sample value for the selected argument.

Output Arguments Node

Lets you see the properties of all output arguments.

User interface elements are described below:

UI Elements	Description
	Opens the pane for editing the selected argument's value.
Name	The name of the operation (read-only)
Negative Testing	<p>Enables Negative Testing. Confirms that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued a SOAP Fault, and not a SOAP result response.</p> <p>Select an Expected Response.</p> <ul style="list-style-type: none"> ➤ 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. <p>For details, see "Negative Testing Overview" on page 1040.</p> <p>Note: Only available with a Service Test license.</p>
Argument List	A list of the output arguments and the corresponding parameters storing the values.

<Output Argument Name> Node

Lets you specify a parameter for storing the value of the output argument.

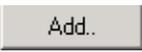
User interface elements are described below:

UI Elements	Description
	Opens the Add Array Elements Dialog Box for adding a new array element to the output argument (only visible when selecting a parent node in the output array). For details, see the "Add Array Elements Dialog Box" on page 956.
	Removes the selected array element in the output argument (only visible when selecting a parent node in the output array).
Name	The name of the output argument or array.
Save returned value in parameter	Saves the value of the selected argument to a parameter. To specify a custom parameter name, modify the default Param_<arg_name> in the Parameter field.
Nil	Sets the value of the current argument to nil=true .
XML (for arrays only)	XML code containing the argument values. To parameterize this value, click the ABC icon (only available for arrays).

Output Attachments Node

Lets you set the properties for output attachment parameters. This is only visible if you enabled output attachments in the "<Operation Name> Node" on page 946.

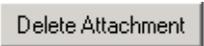
User interface elements are described below:

UI Elements	Description
	Adds a new index-based output argument. This is only available when choosing Save Attachments by Index .
Save All Attachments	Saves all output attachments to a parameter with the following properties: <ul style="list-style-type: none"> ▶ Content. An editable name for the parameter storing the attachment ▶ Content-type. The type of parameter (read-only). ▶ Content-ID. A unique ID for the parameter (read-only).
Save Attachments by Index	Saves the output attachments to index-based parameters. To set the index, select one of the parameters and modify the index number in the right pane.

<Output Argument> Node

Lets you set the properties for output attachment parameters. This is only visible if you enabled output attachments in the "<Operation Name> Node" on page 946.

User interface elements are described below:

UI Elements	Description
	Deletes the selected attachment parameter. If you have saved the attachments by index, it only removes the selected item.
Index	An index number for the parameter. This field is only enabled when you select Save Attachments by Index in the "Output Attachments Node" on page 953.

UI Elements	Description
Content	An editable name for the parameter storing the attachment
Content-type	The content type of parameter (read-only).
Content-ID	A unique content ID for the parameter (read-only).

Add Input Attachment Dialog Box

This page enables you to add input attachments to your Web requests.

To access	Click  Add Service Call and select the top node—the Operation name. Select Add to Request (input) in the Attachments section.
Important information	You must import a service before adding an attachment to a service call. For background information, see "Web Service Call Attachments" on page 916.
Relevant tasks	"Add a new service call- optional" on page 929

User interface elements are described below:

UI Elements	Description
Take data from	<p>The location of the data.</p> <ul style="list-style-type: none"> ▶ File. The file location: <ul style="list-style-type: none"> ▶ Absolute Path: The full path of the file. Note that this file must be accessible from all machines running the script. ▶ Relative Path: (recommended) A file name. Using this method, during replay, VuGen searches for the attachment file in the script's folder. To add it to the script's folder, select File > Add Files to Script and specify the file name. ▶ Parameter. The name of a parameter containing the data.
Content-type	<p>The content type of the file containing the data. The Detect Automatically option instructs VuGen to automatically determine the content type. The Value box accepts manual entries and provides a dropdown list of common content types.</p>
Content-id	<p>A unique identifier for the attachment. By default, VuGen generates this automatically. Optionally, you can specify another ID in the Value box.</p>

Add Array Elements Dialog Box

This page enables you to add elements to an argument array with an identical structure to the existing array. This is available for both Input and Output arguments.

For Input elements, you can base the new array's values on an existing element.

To access	Click  Add Service Call . Select an argument node that is an array.
Important information	You must have an array in your argument tree in order to view this dialog box.
Relevant tasks	"Add a new service call- optional" on page 929

User interface elements are described below:

UI Elements	Description
Name	The name and index of the array's parent node.
Start Index	The index from which to add new array elements.
Elements	The number of identical array elements to add to your argument tree.
Copy values from index	Creates the new array elements with the values of a specific array element (only available for Input arguments).

Process Base64 Data - Simple Data Dialog Box

This dialog box enables you to set the encoding options for your simple base64 data.

To access	For simple, non-complex Base64 values: <ul style="list-style-type: none"> ➤ Select an input argument in the Web Service Call Properties of Base64 type. ➤ Select the Value option. ➤ Choose Embed encoded text. ➤ Click the Browse button.
Important information	For a complex array, use the "Process Base64 Data - Complex Data Dialog Box" on page 958.
Relevant tasks	"Add a new service call- optional" on page 929

User interface elements are described below:

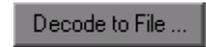
UI Elements	Description
	Allows you to save the decoded text to a file.
Text to encode	For complex data, use the "Process Base64 Data - Complex Data Dialog Box" on page 958.
Encoding Options	A list of encoding methods. Default: Unicode (UTF-8)
Encoded data	The encoded version of the data from the Text to encode pane.

Process Base64 Data - Complex Data Dialog Box

This dialog box enables you to set the encoding options for your complex base64 data.

To access	<p>For complex Base64 values:</p> <ul style="list-style-type: none"> ▶ Select a complex input argument in the Web Service Call Properties, of Base64 type. ▶ Select the Value option click the Parameter icon. ▶ Replace the value with a parameter. ▶ Right-click the Parameter icon in the Value box and select Parameter Properties. ▶ Click the Edit Data button. ▶ In the desired values set column, click the B64 button. <p>Note: Accessible also from the Checkpoint and Service Emulation data grids.</p>
Important information	For simple, non-complex data, use the "Process Base64 Data - Simple Data Dialog Box" on page 957.
Relevant tasks	"Add a new service call- optional" on page 929

User interface elements are described below:

UI Elements	Description
	Encodes the specified file.
	Allows you to save decoded data to a file. This is usually data obtained during replay.

UI Elements	Description
File	<p>Encode the file by reference or its contents.</p> <ul style="list-style-type: none"> ▶ File path. The file to encode. ▶ Link to file. References the file containing the values. If cleared, it uses the content of the specified file. It copies the content to the script folder. <p>Tip. For text exceeding 10KB, enable Link to file.</p>
Text	<p>Encodes the specified text string.</p> <ul style="list-style-type: none"> ▶ Text to encode. The Base64 text to encode. As you type the text, VuGen encodes it in the Encoded data pane. ▶ Encoding Options. A list of encoding methods. The default is Unicode (UTF-8).

Aspect Reference

This section includes:

- ▶ Select Services Screen on page 941
- ▶ Start Recording Dialog Box on page 942
- ▶ Import SOAP Dialog Box on page 944
- ▶ New Web Service Call Dialog Box on page 945
- ▶ Add Input Attachment Dialog Box on page 954
- ▶ Add Array Elements Dialog Box on page 956
- ▶ Process Base64 Data - Simple Data Dialog Box on page 957
- ▶ Process Base64 Data - Complex Data Dialog Box on page 958
- ▶ Generate Test Generation Wizard on page 962
- ▶ Specify Services Screen on page 967
- ▶ Specify Traffic Information Screen on page 968
- ▶ SSL Configuration Dialog Box on page 969

 **Aspects List**

The following table lists the available testing aspects:

Aspect Name	Description
Positive Testing	Generates a full positive test that checks each operation of the service.
Standard Compliance	Checks the service's compliancy with industry standards such as WS-I and SOAP.
Service Interoperability	<p>Tests the interoperability of the service's operations with all supported Web Services toolkits.</p> <p>Contains the following sub-aspects:</p> <ul style="list-style-type: none"> ▶ .NET Framework. Tests that the services are fully interoperable with .NET Framework WSE 2 Toolkit by calling all of its operations with default/expected values. ▶ Axis/Java Based Web Services. Tests that the services are fully interoperable with Axis 1.3 Web Services Framework by calling all of its operations with default/expected values.
Security Testing	<p>Tests service security. Contains the following sub-aspects:</p> <ul style="list-style-type: none"> ▶ SQL Injection Vulnerability. Checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters. ▶ Cross-site Scripting (XSS). Attempts to hack the service by injecting code into a Web site that will disrupt its functionality.

Aspect Name	Description
Boundary Testing	<p>Using the negative testing technique, creates tests to manipulate data, types, parameters, and the actual SOAP message to test the service to its limits.</p> <p>Contains the following sub-aspects:</p> <ul style="list-style-type: none"> ▶ Extreme Values. Provides invalid data types to the services and verifies they are not accepted. ▶ Null Values. Provides NULL parameters to the services to verify they are not accepted.
Performance Testing	<p>Contains the following sub-aspects:</p> <ul style="list-style-type: none"> ▶ Stress Testing. Tests the maximum load that can be placed on the application. ▶ Overload Sustainability Testing. Tests how well the hardware allocated for the application can support the number of anticipated users. ▶ Volume Testing. Tests that the system can handle a massive data entry. ▶ Longevity Test. Tests that the system can sustain a consistent number of concurrent Vusers executing transactions using near-peak capacity, over a minimum 24-hour period. ▶ Scalability Testing. Repeated stress, overload, volume, and longevity tests with different server or network hardware configurations.

Test Generator Wizard User Interface

This section includes (in alphabetical order):

- ▶ Select Services Screen on page 941
- ▶ Start Recording Dialog Box on page 942
- ▶ Import SOAP Dialog Box on page 944
- ▶ New Web Service Call Dialog Box on page 945
- ▶ Add Input Attachment Dialog Box on page 954

- ▶ Add Array Elements Dialog Box on page 956
- ▶ Process Base64 Data - Simple Data Dialog Box on page 957
- ▶ Process Base64 Data - Complex Data Dialog Box on page 958
- ▶ Generate Test Generation Wizard on page 962
- ▶ Specify Services Screen on page 967
- ▶ Specify Traffic Information Screen on page 968
- ▶ SSL Configuration Dialog Box on page 969

Generate Test Generation Wizard

This wizard enables you to create requirements and/or tests for your service.

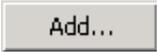
To access	File > New. Select SOA Test Generator in the New Virtual User Dialog Box and click Create .
Relevant tasks	"How to Generate a Test Automatically" on page 932
Wizard map	This wizard contains: Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page > Generate Page > Finish Page

Services Page

This wizard page enables you to indicate which entries to create: requirements and tests, tests only, or requirements only.

Important information	General information about this wizard is available here: "Adding Web Service Script Content Overview" on page 910.
Wizard map	This wizard contains: Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page > Generate Page > Finish Page

User interface elements are described below:

UI Elements (A-Z)	Description
	Opens the Import Service dialog box. For details, see "Import Service Dialog Box" on page 1008.
	Removes the selected service.
<services list>	A list of the imported services.
Service Details	<ul style="list-style-type: none"> ▶ WSDL Location. The location from which the WSDL was imported. ▶ Description. The description of the service from the WSDL.

Select Testing Aspects Page

This wizard page enables you to select testing aspects for which to create the requirement and tests.

Important information	General information about this wizard is available here: "Adding Web Service Script Content Overview" on page 910.
Wizard map	<p>This wizard contains:</p> <p>Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page > Generate Page > Finish Page</p>

User interface elements are described below:

UI Elements (A-Z)	Description
<input type="checkbox"/>	Clear aspect. Excludes the aspect.
<input checked="" type="checkbox"/>	Select Aspect. Includes the aspect. If you select a parent, all of its children are included. Tip: To select all aspects, select the parent Testing Aspects node.
<covered aspects>	A tree view of all the aspects available for your service. Use the Expand <input type="checkbox"/> and Collapse <input type="checkbox"/> buttons to show or hide children aspects.

Select Locations Page

This wizard page enables you to select the storage location for the generated tests.

Important information	General information about this wizard is available here: "Adding Web Service Script Content Overview" on page 910.
Wizard map	This wizard contains: Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page > Generate Page > Finish Page

User interface elements are described below:

UI Elements (A-Z)	Description
	Opens the Application Lifecycle ManagementConnection - Server Connection dialog box. For details, see "HP ALM Connection Dialog Box" on page 253.
	<ul style="list-style-type: none"> ▶ For Application Lifecycle Managementoutput: Locate a target folder in the ALM repository. ▶ For file system output: Locate a target folder on the file system.
Output to Application Lifecycle Management	<p>Store the scripts in the Application Lifecycle Managementrepository.</p> <ul style="list-style-type: none"> ▶ Name. The name of the sub folder in which to store the scripts. ▶ Location. The parent folder. The Name will be a sub folder of this location.
Output to local file system	<p>Store the scripts on a file system.</p> <ul style="list-style-type: none"> ▶ Name. The name of the sub-directory in which to store the scripts. ▶ Location. The parent directory. The Name will be a sub-directory of this location.

Generate Page

This wizard page enables you to indicate which entries to create: requirements and tests, tests only, or requirements only.

Important information	General information about this wizard is available here: "Adding Web Service Script Content Overview" on page 910.
Wizard map	<p>This wizard contains:</p> <p>Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page> Generate Page > Finish Page</p>

User interface elements are described below:

UI Elements (A-Z)	Description
	Generates the listed scripts, based on the selected aspects.
<test list>	A list of the scripts to generate. To exclude an item, clear its checkbox.

Finish Page

This wizard page enables you to indicate which entries to create: requirements and tests, tests only, or requirements only.

Important information	General information about this wizard is available here: "Adding Web Service Script Content Overview" on page 910.
Wizard map	This wizard contains: Welcome > Services Page > Select Testing Aspects Page > Select Locations Page > Summary Page > Generate Page > Finish Page

User interface elements are described below:

UI Elements (A-Z)	Description
	Opens the selected scripts in VuGen.
<generated scripts>	A list of the scripts that were generated. To exclude an item, clear its checkbox.

Analyze Traffic User Interface

This section includes:

- Select Services Screen on page 941

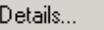
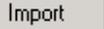
- Start Recording Dialog Box on page 942
- Import SOAP Dialog Box on page 944
- New Web Service Call Dialog Box on page 945
- Add Input Attachment Dialog Box on page 954
- Add Array Elements Dialog Box on page 956
- Process Base64 Data - Simple Data Dialog Box on page 957
- Process Base64 Data - Complex Data Dialog Box on page 958
- Generate Test Generation Wizard on page 962
- Specify Services Screen on page 967
- Specify Traffic Information Screen on page 968
- SSL Configuration Dialog Box on page 969

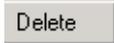
Specify Services Screen

This wizard screen enables you to select Web services to associate with your traffic-based script.

To access	Analyze Traffic button
Relevant tasks	"Record a session - optional" on page 928

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

UI Elements (A-Z)	Description
	Opens the Manage Services dialog box for providing further information about the service. For more information, see "Manage Services Dialog Box" on page 1003.
	Opens the Import Service dialog box. For more information, see "Import Service Dialog Box" on page 1008.

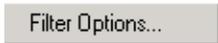
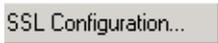
UI Elements (A-Z)	Description
	Removes the selected service from the list.
<services list>	A list of the available services: <ul style="list-style-type: none"> ➤ Service Name. The native name of the service. ➤ WSDL Location. The source of the WSDL.

Specify Traffic Information Screen

This wizard screen enables you to specify the capture file for the incoming or outgoing traffic.

To access	Analyze Traffic button, Next
Relevant tasks	"Record a session - optional" on page 928

User interface elements are described below:

UI Elements (A-Z)	Description
	Browse. Allows you to select a capture file to import.
	Opens the Traffic Filters node in the Recording Options dialog box. This allows you to specify which IP addresses to include or exclude from the script. For details, see "Recording Options" on page 307.
	Opens the "SSL Configuration Dialog Box" on page 969 which allows you to add SSL certificates to analyze traffic from a secure server.
Capture file	The name of a capture file containing the server traffic, usually with a cap extension.
Incoming Traffic	The IP address and port of the server whose incoming traffic you want to examine.

UI Elements (A-Z)	Description
Outgoing Traffic	The IP address of the server whose outgoing traffic you want to examine.
Record into action	The section into which you want to create the script: vuser_init , Action , or vuser_end . For actions you want to repeat, use the Action section. For initialization steps, use vuser_init .

SSL Configuration Dialog Box

This dialog box enables you to configure the SSL certificate for your traffic file.

To access	Analyze Traffic button, Next , 
Relevant tasks	"Record a session - optional" on page 928

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

UI Elements (A-Z)	Description
	Add Certificate. Adds a new line to the certificate list.
	Delete. Removes the selected certificate.
<certificate list>	The properties of the certificate. Specify the following: <ul style="list-style-type: none"> ▶ IP. The URL or IP address of the machine hosting the certificate file. ▶ Port. Port of machine hosting the certificate file. ▶ File. The path of the certificate file (with a pem extension) containing the private key. Use the Browse button to locate the file. ▶ Password. A password for decrypting the certificate file.

36

Web Services - Managing Services

This chapter includes:

Concepts

- ▶ Managing Services Overview on page 972
- ▶ Tasks on page 991
- ▶ XML Editing on page 978
- ▶ XML Validation Overview on page 980
- ▶ XML Queries on page 990

Tasks

- ▶ How to Add and Manage Services on page 991
- ▶ Reference on page 1003
- ▶ How to Validate the XML on page 994
- ▶ How to Edit the Schema on page 996
- ▶ How to Edit Values in the XML Grid on page 998
- ▶ How to Build an XML Query on page 1001

Reference

- ▶ Service Management User Interface on page 1003
- ▶ SOA Tools User Interface on page 1011

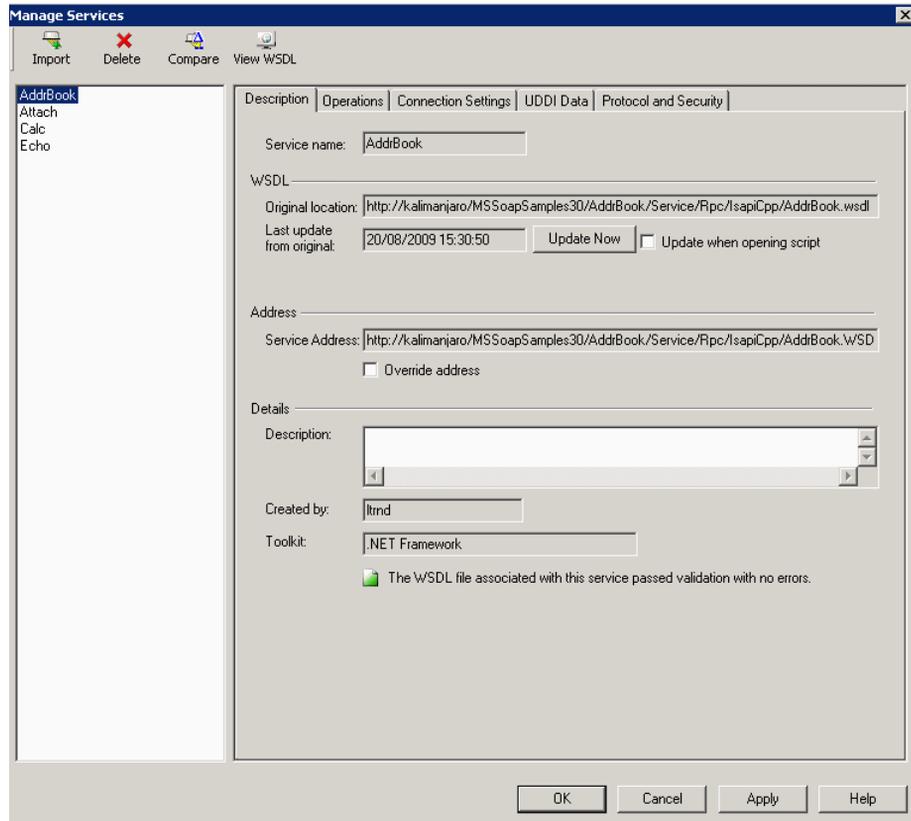
Concepts

Managing Services Overview

The Service Management window lets you manage a list of service entries for the current script. You can view and set the properties of each service entry.

You add service entries to the list by importing WSDL files. When you add a WSDL to the list, VuGen creates a working copy that it saves with the script—it is not global. Therefore, for each script that you create, you must import the desired WSDL files.

The Manage Services window provides quick access buttons for importing, deleting, and comparing services.



The tabs provide you with the ability to set the WSDL properties.

Description

The **Description** tab displays information about the service:

- **Original location.** The original source of the WSDL file (read-only).
- **Service name.** The name of the Web Service (read-only).
- **Last update from original.** The last date that the local copy was updated from the original source (read-only). You can update the version manually or retrieve it automatically each time you reopen the test.

- **Service address.** An endpoint address to which the request is sent. If required, you can override the endpoint specified in the WSDL file.

This is useful for implementing emulated services, available with a Service Test license. VuGen with Service Test uses the **override address** as the targetAddress for the Web Service call. This overriding affects all Web Service calls. To use a different target address for a particular Web Service call, you specify it in that step's properties. For details, see "How to Create an Emulated Service – Workflow" on page 1139.

- **Created by.** The name of the user who originally imported the service (read-only).
- **Toolkit.** The toolkit associated with the script. You set this before importing the first WSDL file (read-only).

Operations

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

Description Operations Connection Settings UDDI Data Protocol and Security			
Operation Name	Port Name	Used In Script	
AddAddr	AddrBookSoapPort	No	
ChangeAddr	AddrBookSoapPort	No	
DeleteAddr	AddrBookSoapPort	No	
Export	AddrBookSoapPort	No	
GetAddr	AddrBookSoapPort	No	
GetNames	AddrBookSoapPort	No	
Import	AddrBookSoapPort	No	

Connection Settings

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

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

- SSL
- Basic and NTLM authentication
- Kerberos for the .NET toolkit

For more information about setting the connection information while importing the WSDL, see "Import a service" on page 991.

UDDI Data

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

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

Protocol and Security Settings

The Protocol and Security Settings tab shows the details of the security scenario applied to the script. If you did not choose a scenario, it uses the default <no scenario>. For more information, see "Set a security scenario - optional" on page 992.

This section also includes:

- "Importing Services" on page 976
- "Comparison Reports" on page 977

Importing Services

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

The Import mechanism requires the following information:

- ▶ **Source.** The source of the WSDL: URL, File, UDDI, or Application Lifecycle Management. UDDI is a universal repository for services (Universal Description, Discovery, and Integration). Service brokers register and categorize published Web Services and provide search capabilities. The UDDI business registry is an example of a service broker for WSDL-described Web Services.
- ▶ **Location.** the path or URL of the WSDL, entered manually or by browsing.
- ▶ **Toolkit.** The toolkit to permanently associate with all services in the script for all subsequent imports and replays (only available for the first service added to the script). The toolkit setting instructs VuGen to send real client traffic using an actual toolkit—not an emulation.

VuGen supports the .NET Framework with WSE 2 version SP3 and Axis/Java based Web Services Framework toolkits. VuGen imports, records, and replays the script using the actual .NET or Axis toolkit. By default, VuGen uses automatic detection to determine the most appropriate toolkit.

- ▶ **Connection Settings.** Authentication or proxy server information. This setting is useful for WSDLs residing on secure servers, or WSDLs that must be accessed via a proxy server.

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

For task details, see "How to Add and Manage Services" on page 991.

Comparison Reports

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

You can configure the comparison settings, indicating which items to ignore during the comparison. For more information, see the "XML/WSDL Comparison Dialog Box" on page 1011.

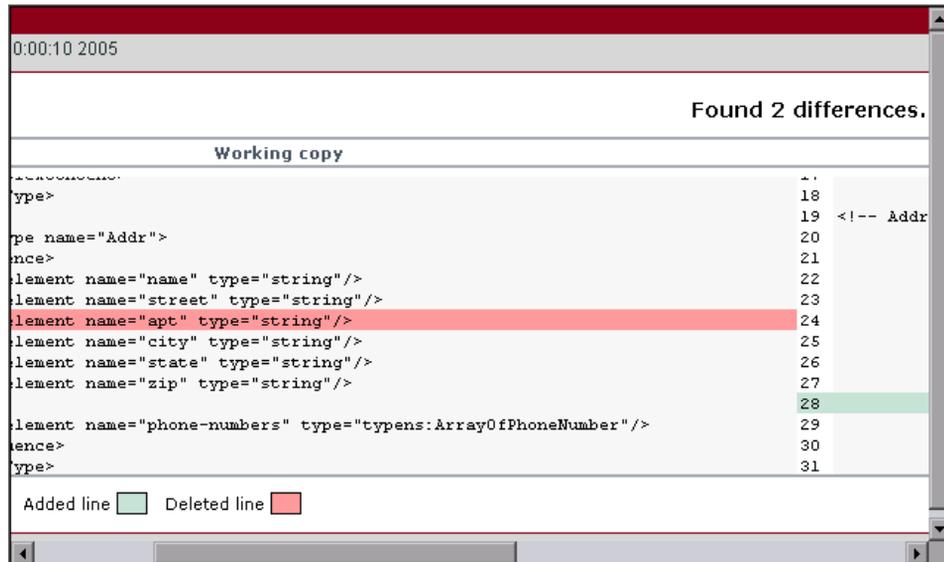
In WSDL Comparison reports, there are two columns— **Working Copy** and **Original File**. The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

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.



0:00:10 2005

Found 2 differences.

Working copy	
type>	18
	19 <!-- Addr
pe name="Addr">	20
nce>	21
lement name="name" type="string"/>	22
lement name="street" type="string"/>	23
lement name="apt" type="string"/>	24
lement name="city" type="string"/>	25
lement name="state" type="string"/>	26
lement name="zip" type="string"/>	27
	28
lement name="phone-numbers" type="typens:ArrayOfPhoneNumber"/>	29
ence>	30
ype>	31

Added line ■ Deleted line ■

Web Reference Analyzer

Many WSDL files reference other files such as XSD and other XML files. Before running a script, you may want to determine what these files are and if they are available.

VuGen's WSDL Reference Analyzer checks the WSDL for dependencies, and lists them in the WSDL Reference Analyzer window and in a log file.

The Analyzer places the WSDL and its dependent files in a zip archive file. It saves the dependency information to a log file, listing its path in the Analyzer window. For task details, see "How to Analyze WSDL Dependencies" on page 993

XML Editing

The XML editor lets you view and edit XML and XSD schemas. This editor is only available with a Service Test license. For details, contact HP Support.

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

Schema	Set 1
- Addr	
REC name	John Smith
REC street	3 Acorn Lane
REC city	NIL Phoenix
REC state	AZ
REC zip	NIL 65354
REC zip4	3333
phonenumbers	
PhoneNumber [..]	
PhoneNumber[1]	NIL

While you can use the XML Editor as a standalone editor to format your XML code, it is also integrated with other VuGen with 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.

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			
phonenumber			
PhoneNumber [..]			
PhoneNumber[1]	NIL	NIL	NIL

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

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

For task details, see "How to Edit the Schema" on page 996.

XML Validation Overview

When you run a Web Service, it transfers data requests and responses to and from a server. In Web Services scripts, it is common to use parameters for the data instead of hard-coded values. Each parameter contains the actual XML data being transferred. By using parameters, you can test your Web Service with many different values.

VuGen provides an interface for validating the XML data in the parameters by checking for:

- ▶ well-formed XML
- ▶ correctness of checkpoint values
- ▶ compliance with an XML schema (manual loaded schemas only)

To be well-formed, an XML document must follow more than 100 different rules as defined by the World Wide Web Consortium (W3C). For example, one important rule is that each element must have start and end tags.

Checkpoint validation compares argument values within the XML file with expected values. You manually specify the expected values or load them from an earlier session.

An XML schema represents the interrelationship between the attributes and elements of the XML objects. To check that an XML file complies with its schema, you specify the XML file together with its XSD (XML Schema Definition) file. This check only applies to XSDs that you load manually—not ones that VuGen detects automatically.

This section also includes:

- ▶ "XSD Schema Validation" on page 981
- ▶ "Checkpoints and the Expected Response" on page 982
- ▶ "Validation Results" on page 986
- ▶ "REST Services and XML Validation" on page 989

XSD Schema Validation

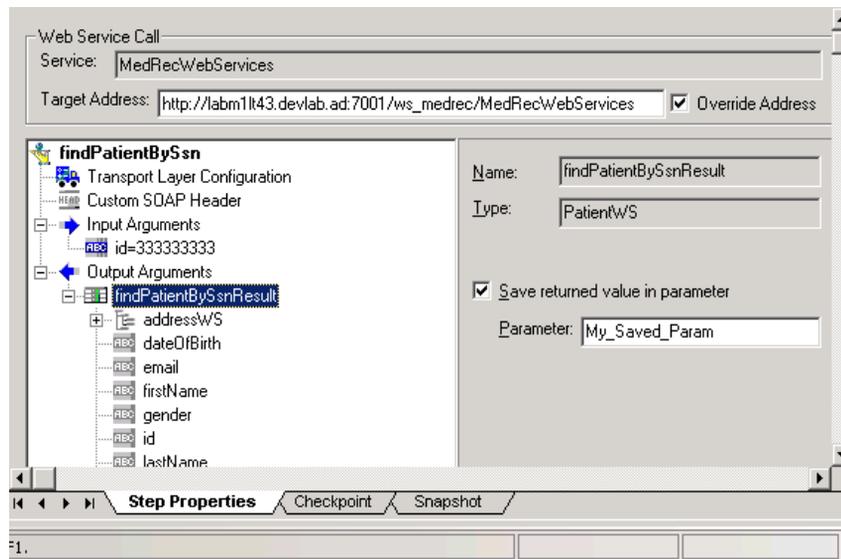
To perform complete XML validation, you select a parameter and indicate its corresponding XSD schema. When specifying a parameter for XML validation, you can use either an input or output parameter.

For input parameters, you use a parameter that you saved earlier. You can also select **Create/Select Parameter** from the **XML Source** dropdown list to create a new one. The parameter type should be XML parameter or a file type parameter where the file's content is XML.

Ideally, this parameter is XML with an ANY type element. By parsing the XSD, VuGen determines whether or not the WSDL is compliant with the schema.

You create output parameters in the Properties tab, by enabling **Save returned value in parameter** while selecting the output value. If you select the top node of the output parameter, VuGen saves the value as an XML parameter.

The following example shows an output array saved to a parameter, **My_Saved_Param**.



For information about defining parameters, see "Parameters" on page 257.

In the XML Validation dialog box, you provide the saved parameter as the **XML Source**.

Checkpoints and the Expected Response

The following sections describe the types of checkpoints and how they help you validate the expected response.

Basic and Advanced Checkpoints

You can perform both basic or advanced validations for your checkpoints.

In basic validation, VuGen looks for exact matches of the value in the **Expected value** column. You can load expected values from a Recorded or Replay snapshot.

Use **Advanced Checkpoints** to validate a checkpoint on non-leaf nodes or to define expected values in terms of a regular expression.

You define the advanced validation values by entering an XPATH query in the **Advanced Checkpoint** section. To obtain the initial XPATH expression, copy it from a row in the **Basic Validation**. Choose **Copy Row XPATH** from the right-click menu and paste the contents in the **Advanced Validation** section.

You can define both basic and advanced validations for the same step.

Note that when you select a non-leaf node, you need to supply all of the XML beneath the node.

In the Vuser script, VuGen indicates an exact match by **Value=** and a regular expression with **Expression=**:

```
BEGIN_CHECKPOINTS,  
    CHECKPOINT, "XPATH=/AddResult[1]", "Value=50"  
    CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",  
END_CHECKPOINTS,
```

XPATH Expressions

Checkpoint expressions support both Nodset and extended XPath syntax.

NodeSet expressions are XPath expressions evaluated to a single XML node or to a set of XML nodes. For example:

```
/a/b/c
/x/y[1]
//a/b
```

VuGen also supports full XPath expressions, such as **count(/a/b/c)** which is evaluated to int, or **count(/a/b/c) < 3** which is evaluated to a boolean value. The XPATH support allows you to do numeric comparisons. For example, to verify whether a result is between 8 and 16, you can use the following expression:

```
number(/a/b/c) > 8 and number(/a/b/c) < 16
```

The Expected Response

Using XML Validation you can check the entire response of your Web Service call. This capability is similar to the standard checkpoints, but it has the added capability of saving the entire response to a single parameter, instead of saving each argument to its own parameter.

To validate your responses, you add an XML validation step after the Web Service call you want to validate (it need not be immediately after the call).

For each Web Service call, VuGen saves the entire XML response to a parameter. This lets you validate the entire response in a single step. The name of the parameter is the Web Service call step name with a **_Response** suffix. For example, if the step name is **GetAddr_101**, the response parameter is **GetAddr_101_Response**.

Note that this parameter is not the actual SOAP response. The SOAP response contains the response values and the entire SOAP envelope. This response while containing the XML values, does not contain the SOAP header or envelope.

Since the response is unique for each Web Service call, you must add separate validation steps for each saved parameter. When you select a response parameter, VuGen automatically loads the XSD, in order to perform a schema validation.

You can also validate an individual argument, whose value you manually saved to a parameter in design time. This parameter will appear in the **XML Source** dropdown.

Validation Comparison Operators

You use checkpoints in XML validation to check for expected values in the XSD using the following comparison operators:

- **Equals.** match the exact text.
- **Does not equal.** Does not match the text.
- **Contains.** match part of the text.
- **Starts With.** a match if the returned text begins with the following phrase.
- **Ends With.** a match if the returned text ends with the following phrase.
- **Reg. expression.** matches the string pattern using regular expression syntax

For numerical elements, you indicate one of the standard numerical relationships, such as equal, not equal, greater than, less than, and so forth.

For information on defining advanced queries using regular expressions, see "XPath Expressions" on page 982.

Optional Elements

The XML Validation tool lets you check for the presence of an optional value. In some cases, you need to check that a value is present, while in others you need to verify that a value is not present.



When you include the optional element, by filling in the icon, replay checks that the service returned a value. When you exclude the optional element, by clearing the optional icon, replay verifies that the service did not return a value.



In the following example, the optional element was included and given the value of **abcd**.

XML Check Points	Validate	Expected Value
Choice [..]		
Choice[1]		
Ad		
Choice [..]		
Choice[1]		
<input type="radio"/> ABC ImageUrl	<input type="checkbox"/>	Equals ▾
<input type="radio"/> ABC NavigateUrl	<input type="checkbox"/>	Equals ▾
<input checked="" type="radio"/> ABC Keyword	<input checked="" type="checkbox"/>	Does not equal ▾ abcd
<input type="radio"/> 123 Impressions	<input type="checkbox"/>	Equals ▾
<input type="radio"/> ABC AlternateText	<input type="checkbox"/>	Equals ▾
<input type="radio"/> 123 Width	<input type="checkbox"/>	Equals ▾
<input type="radio"/> 123 Height	<input type="checkbox"/>	Equals ▾

During replay, VuGen checks for a value for the element. If it does not detect any value during replay, VuGen issues an error message indicating that it did not find a value for an element that was included. You can see detailed information in the Test Results report. For more information, see "Run the script and view the results" on page 996.

In the following example, VuGen checks that the server did not return a value. If it finds a value for the element, an error message indicates that VuGen detected a value for an element that should have been excluded.

XML Check Points	Validate	Expected Value
Choice [..]		
Choice[1]		
Ad		
Choice [..]		
Choice[1]		
<input type="radio"/> ABC ImageUrl	<input type="checkbox"/>	Equals ▾
<input type="radio"/> ABC NavigateUrl	<input type="checkbox"/>	Equals ▾
<input checked="" type="radio"/> ABC Keyword	<input checked="" type="checkbox"/>	Does not equal ▾ abcd
<input type="radio"/> 123 Impressions	<input type="checkbox"/>	Equals ▾
<input type="radio"/> ABC AlternateText	<input type="checkbox"/>	Equals ▾
<input type="radio"/> 123 Width	<input type="checkbox"/>	Equals ▾
<input type="radio"/> 123 Height	<input type="checkbox"/>	Equals ▾

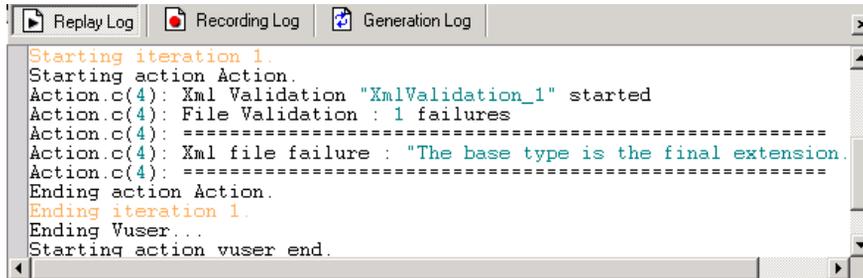
For both types of testing—inclusion and exclusion—you need to select the checkbox in the **Validate** column. Otherwise, the Validation tool will ignore the element.

Validation Results

By adding an XML Validation step to your script, VuGen performs the validations in runtime. You can determine the validation results in several ways:

Replay Log

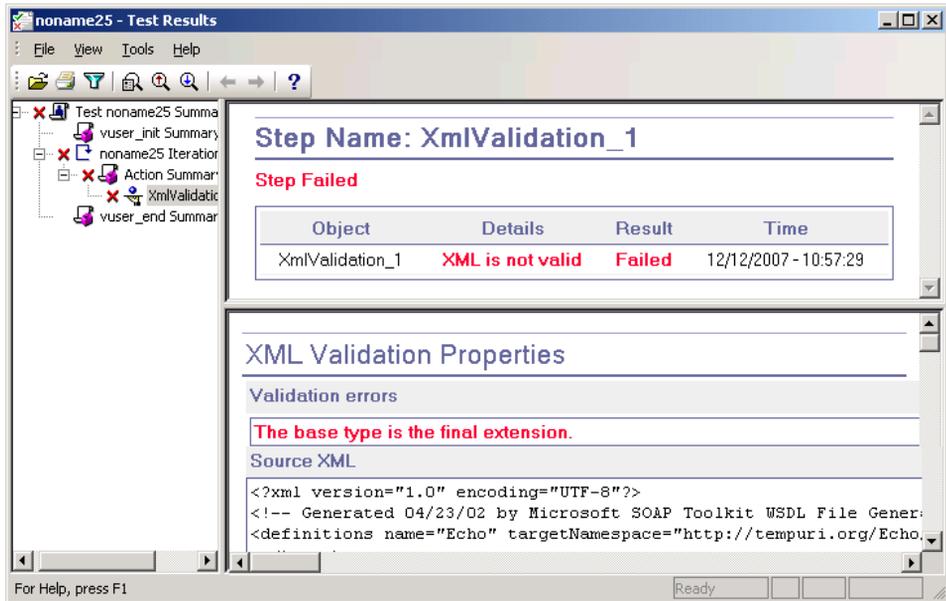
During replay, VuGen displays the results in the Replay log. We recommend that you enable the Extended Log Run-Time settings to obtain more details about the validation.



```
Replay Log | Recording Log | Generation Log
Starting iteration 1.
Starting action Action.
Action.c(4): Xml Validation "XmlValidation_1" started
Action.c(4): File Validation : 1 failures
Action.c(4): =====
Action.c(4): Xml file failure : "The base type is the final extension.
Action.c(4): =====
Ending action Action.
Ending iteration 1.
Ending Vuser...
Starting action vuser end.
```

Test Results Report

The Test Results report shows the results of the validation step, along with the XML source code. To open the report, select **View > Test Results**.



Return Values

In addition to viewing the Replay log, you can evaluate the return values for `soa_xml_validate` to determine the outcome of the validation. The following table lists the possible return values:

Value	Description
0	Validation succeeded. XML is well-formed, compliant, and checkpoints are in order.
1	The parameter is empty or does not exist.
2	The XML is not well-formed.
3	The XML is well-formed, but not the XSD.
4	The XML does not comply with the XSD.

Value	Description
5	The values in the XML string do not match the values specified in the checkpoints' XPATH queries.
6	The XSD file could not be found.
7	An error, other than the ones listed in this table, occurred, such as an internal or unknown error.
8	The XSD file is not well-formed and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.
9	XML failed validation against the XSD file and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.

For more information about the function and its return values, see the *Online Function Reference* (**Help > Function Reference**).

REST Services and XML Validation

You can use the XML validation for non-SOAP Web Services, such as REST. The following example illustrates the collaboration between a REST Web Service and XML Validation.

```
// Save the content between the commas to a Design Run type parameter
web_reg_save_param("WCSPParam_Text1",
    "LB=",
    "RB=",
    "RelFrameId=1",
    "Search=Body",
    "IgnoreRedirections=Yes",
    LAST);

//The service request submitted as a URL
web_url("xml",
    "URL=http://ecs.amazonaws.com/onca/
xml?Service=AWSECommerceService&Operation=ItemSearch&AWSAccessKeyId=12
3456789ABCD&SearchIndex=Books&Keywords=Cortisol&Author=Tolstoy",
    "Resource=0",
    "RecContentType=text/xml",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    LAST);

//Use the parameter's value in the validation statement.
soa_xml_validate ("StepName=XmlValidation_1",
    "Snapshot=t3e9876543214.inf",
    "XML={WCSPParam_Text1}",
    "StopOnValidationError=0",
    "XSD=C:\\Bugs\\67571_Rest\\AWSECommerceService.xsd",
    BEGIN_CHECKPOINTS,
    ...
    END_CHECKPOINTS,
    LAST);
```

To use a POST, PUT, or DELETE action, use `web_custom_request` as described in the *Online Function Reference*.

XML Queries

You can query an XML tree in order to locate and examine a specific element and value.

To search an XML file, you use a query in the standard XPATH syntax. To build a valid XPATH query, use the built-in Query Builder. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

Tasks

How to Add and Manage Services

This task describes how to create a list of services that you can call from your test. Using the Manage Services window, you import services and configure their settings.

This task includes the following steps:

- "Open the Manage Services dialog box" on page 991
- "Import a service" on page 991
- "Get to know the WSDL" on page 992
- "Check for WSDL updates - optional" on page 992
- "Override the service address- optional" on page 992
- "Set a security scenario - optional" on page 992

Open the Manage Services dialog box

Select **SOA Tools > Manage Services** or click the toolbar button to open the Manage Services dialog box.

Import a service

Click **Import**. In the Import Service dialog box, select a WSDL source and browse to the location.

For **URL** type imports, the Browse button opens a new browser. Navigate to the WSDL and then close the browser. This action places the URL in the location box. For details, see the "Import Service Dialog Box" on page 1008.

If your service requires authentication or uses a proxy, configure these settings before importing the WSDL. Expand the Import Services dialog box and click **Configure**. For details, see the "Connection Settings Dialog Box" on page 1007.

Repeat this step for all the services you want to include in your test.

Get to know the WSDL

Familiarize yourself with the WSDL. View its details as described in the "Description Tab" on page 1005.

Click **View WSDL** to open the locally saved WSDL file in Internet Explorer and study its structure.

Check for WSDL updates - optional

Use the Comparison tool to check that the WSDL did not change since your last import or update.

First, set the comparison options. Click **SOA Tools > SOA Settings > XML/WSDL Comparison**. Specify what differences to ignore. For details, see "XML/WSDL Comparison Dialog Box" on page 1011.

In the Manage Services window, click **Compare** to open a report comparing the working copy of the WSDL with the one at the original location.

If you discover changes in the Comparison report, click **Update Now** to retrieve the latest version of the WSDL from its source.

Override the service address- optional

View the address in the **Service Address** box. This is the default endpoint address as retrieved from the WSDL. If you want to override it, select **Override address** and type in an alternate endpoint address for the service requests.

To return to the default address, clear the **Override address** option. For details, see the "Description Tab" on page 1005.

Set a security scenario - optional

Click the **Protocol and Security** tab to use WS-Security or another type of a security scenario.

For more information, see Chapter 38, "Web Services - Security."

How to Analyze WSDL Dependencies

This task describes how to use the Reference Analyzer to determine WSDL dependencies. For user interface details, see "WSDL Reference Analyzer Dialog Box" on page 1014.

This task includes the following steps:

- "Open the Reference Analyzer" on page 993
- "Check for WSDL updates - optional" on page 992
- "Begin the analysis" on page 994
- "View the log" on page 994

1 Open the Reference Analyzer

Select **SOA Tools > WSDL Reference Analyzer**.

2 Select a source and target

In the **Select WSDL file** box, indicate the location of the WSDL you want to analyze.

In the **Output file path** box, indicate a location for the zip file.

3 Begin the analysis

Click **Start Analyzing**. The Analyzer lists all of the dependencies in the output window along with their paths.

4 View the log

View the results in the log window. To clear the results and perform another analysis, click **Clear Log**.

How to Validate the XML

This task describes how to validate the XML to be well-formed and verify the expected responses. For concept details, see "XML Validation Overview" on page 980.

This task includes the following steps:

- "Open the XML Validation tool" on page 995
- "Select a parameter" on page 995
- "Enable checkpoints" on page 995
- "Set the expected values" on page 995
- "Set advanced checkpoints - optional" on page 995

1 Open the XML Validation tool

Select a test step (or Web Service call in Script view) and click the **Validate XML** button on the upper toolbar. For user interface details, see "Validate XML Dialog Box" on page 1012.

2 Select a parameter

In the XML Source box, choose a response parameter from the drop down list. For the complete response of the Web Service call, choose the response parameter—the name of the Web Service call step with a `_Response` suffix. If you manually saved a response argument for an individual parameter, you can select it too.

3 Enable checkpoints

Select the **Checkpoints** option if it is not already selected. In the **Validate** column, check the arguments that you want to validate. Click **Select All** to enable validation for all of the arguments and their children.

4 Set the expected values

- a** Place the mouse over the argument name to discover its data type.
- b** Select a comparison operator, such as **Equals**, **Does not equal**, and so forth.
- c** Specify an expected value. Certain data types provide an interface for selecting values, such as the Boolean and Date types.
- d** To parameterize a value, click the **ABC** icon and create a new parameter.

5 Set advanced checkpoints - optional

- a** Select the **Advanced Checkpoints** option.
- b** Copy an XPath expression from an existing argument. For example, in the Checkpoint grid, select an argument value and select **Copy XPATH** from the shortcut menu.
- c** Paste the XPath expression into the **XPATH Query** column. For more information, see "XPath Expressions" on page 982.

- d Select a Validation Method: **Contains**, **Reg.Expression**, or **Exact Phrase**.
- e Specify an expected value.

6 Complete the validation step

Click OK to add the XML Validation step. It appears in your script as `soa_xml_validate`.

7 Prepare for a test run

Set the Extended logging options. Click F4 to open the Run-Time settings. Select the **General > Log** node and select the Extended log option. You do not need to enable any of the sub options. Click OK.

8 Run the script and view the results

- a Click F5 to run the script. View the Replay log to check for errors.
- b View the test results. If the Test Results window did not open automatically, select **View > Test Results**.
- c Expand the XML Validation step. View the status, source XML, and the checkpoint results.

How to Edit the Schema

This task describes how to use the XML editor to load values for the XML and XSD files. The editor is only available with a Service Test license. For user interface details, see "XML Editor Dialog Box" on page 1015.

This task includes the following steps:

- "Open the XML editor" on page 997
- "Load a schema" on page 997
- "Select a root" on page 997
- "Assign values" on page 997
- "Add more value sets - optional" on page 997
- "Edit the XML - optional" on page 998

- "Remove a value set - optional" on page 998
- "Save a column and its values- optional" on page 998

1 Open the XML editor

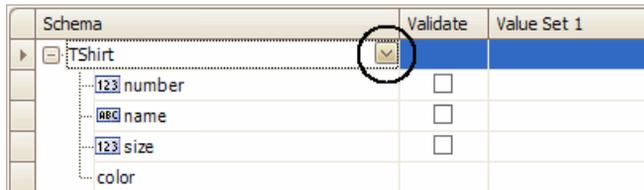
Choose **SOA Tools > XML Editor**. The editor opens.

2 Load a schema

Choose **Schema > Load** and select an XSD file for which you want to create value sets and edit.

3 Select a root

If the schema has multiple root elements, select a root element. Click the small button adjacent to the root name to open the drop down box.



4 Assign values

- Type entries into the **Values** column. To enter values manually, place the mouse over the element's icon to discover its data type. For more information about data types, see "Determine the data type" on page 999.
- To load values from an XML file, Click the **Load XML from file into the selected column** button.



5 Add more value sets - optional



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



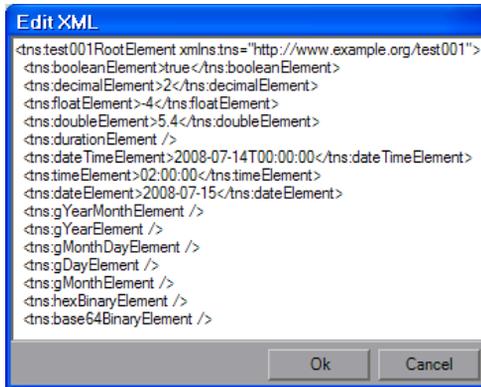
To add a column with identical values to the selected one, click the Duplicate Column button.

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

6 Edit the XML - optional



To edit the actual schema, select one column and click the **Edit XML from the selected column** button. After the editing, click **OK**.



7 Remove a value set - optional



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

8 Save a column and its values- optional



To save the XSD with the values of the current value set, select the column and choose **XML > Save** or click the **Save XML from the selected column** button.

How to Edit Values in the XML Grid

This task describes how to work in the XML grid. The XML grid is integrated into several of the VuGen with Service Test components: XML Editor, Parameterization, XML Validation, and Service Emulation.

Parameterization uses the editor to display parameter elements and values as described in "Parameters" on page 257.

XML validation uses the grid to display the XSD schemas. Service Emulation uses the grid to display rules.

The XML Editor, XML Validation, and Service Emulation tools, are only available with a Service Test license. For details, contact HP Support.

For user interface details, see "XML Editor Dialog Box" on page 1015.

This task includes the following steps:

- "Enter the XML grid" on page 999
- "Determine the data type" on page 999
- "Set the base64 properties" on page 1000
- "Include or exclude optional elements" on page 1000
- "Include or exclude array elements" on page 1000
- "Add or remove array elements" on page 1000
- "Create a structure for <any> elements" on page 1001

Enter the XML grid

Click within an XML grid. The XML grid is integrated into the XML Editor, Parameterization, XML Validation, and Service Emulation. Choose a root element if there are multiple ones.

Determine the data type

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

The grid recognizes element data types and provides an interface for setting the values. For example:

- For an **Int** type, the value cell contains a number scrolling control.
- For boolean, it contains a list box with the values **true**, **false**, **1**, or **0**.
- For a **Date** element, you can open a calendar to select a date.

Set the base64 properties

The schema indicates **Base64** elements with a **B64** button to the right of the value box. Click the button to open the Process Base 64 Data dialog box. For UI details, see "Process Base64 Data - Simple Data Dialog Box" on page 957.

Include or exclude optional elements

To include optional elements, fill in the green triangle adjacent tot the element.

Include or exclude array elements

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

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

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

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

Add or remove array elements

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

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

Create a structure for <any> elements

- To add an array of <any> elements, select the parent <any> element and choose **Add Array Element** from the shortcut menu.
- To add a child <any> element, select the parent element and choose **Add child** from the shortcut menu.
- To add an additional child <any> element, select the child element and choose **Add sibling** from the shortcut menu.
- To provide a name for the <any> element, click the *Rename Element* text, and type in a name.

Schema	Validate	Value Set 1
AnyRootElement001		
Any [...]		
My Name	<input type="checkbox"/>	
Rename element	<input type="checkbox"/>	

- To load values for an <any> element, choose **Load XML** from the shortcut menu.

How to Build an XML Query

This task describes how to use the Query Builder to locate and examine a specific element. For details, see "XML Queries" on page 990.

This feature requires a Service Test license.

This task includes the following steps:

- "Prerequisites" on page 1002
- "Open the Find XML dialog box" on page 1002
- "Specify a query" on page 1002
- "Specify details for the Query Builder" on page 1002
- "Perform the query" on page 1002

1 Prerequisites

Create a script and run it at least once.

2 Open the Find XML dialog box

In the Snapshot tab, Click **Find XPath**. In the Find XML dialog box, select **Request** or **Response**. For user interface details, see the "Find XML Dialog Box" on page 1016.

3 Specify a query

Manually type in an Xpath query or click **Query Builder**.

4 Specify details for the Query Builder

If using the Query Builder, enter the search text in the appropriate boxes.

- ▶ Enable the **Name** section to search for the name of a node or element.
- ▶ Enable the **Namespace URI** section to search for a namespace.
- ▶ Enable the **Text** section to search for the value of the element indicated in the Name box.
- ▶ Enable the **Attributes** section to search for an attribute.
 - ▶ To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.



5 Perform the query

Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the XPath query box. Click **Find Next** to begin the search.

Reference

Service Management User Interface

This section includes (in alphabetical order):

- ▶ Manage Services Dialog Box on page 1003
- ▶ Connection Settings Dialog Box on page 1007
- ▶ Import Service Dialog Box on page 1008
- ▶ Search for Service in UDDI Dialog Box on page 1010
- ▶ XML/WSDL Comparison Dialog Box on page 1011
- ▶ Validate XML Dialog Box on page 1012
- ▶ WSDL Reference Analyzer Dialog Box on page 1014
- ▶ XML Editor Dialog Box on page 1015
- ▶ Find XML Dialog Box on page 1016

Manage Services Dialog Box

This dialog box enables you to manage your WSDLs, provide authentication information, and set a security scenario.

To access	Use one of the following: <ul style="list-style-type: none"> ▶  Manage Services ▶ SOA Tools > Manage Services
Relevant tasks	"Open the Manage Services dialog box" on page 991

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

UI Elements (A-Z)	Description
 Import	Opens the Import Service dialog box.
 Delete	Removes the selected service from the list.
 Compare	Opens the WSDL Comparison Report showing the Working copy and Original copy of the WSDL side-by-side. To set the comparison settings, see XML/WSDL Comparison Dialog Box
 WS-I Validation	Opens the WS-I Validation tool to check the WSDL's compliancy with WS-I. Note: Only visible when a Service Test license is installed.
 View WSDL	Displays the WSDL in a browser.
<WSDL list>	A list of the imported WSDLs.
Description tab	Provides information about the WSDL, its endpoint address, toolkit, and update information.
Operations tab	Lists the operations of the service: Operation Name , Port Name , and Used in Script (Yes or No). Click a column to sort the operations by that column's data. Click it again to reverse the sorting order.
Connection Settings tab	Allows you to provide authentication settings for the machine from which you are importing a service. For more information, see the "Connection Settings Tab" on page 1006. Note: This only applies to URL and UDDI type imports.

UI Elements (A-Z)	Description
UDDI Data	The UDDI server, UDDI server, and service key. For more information, see the "UDDI Data Tab" on page 1007
Protocol and Security tab	Allows you to view and set a security scenario for your Web Service calls. For more information, see below.

Description Tab

The following elements are displayed in the **Description** tab:

UI Elements (A-Z)	Description
	Loads the latest version of the WSDL from its original location.
Created By	The name with which you logged in. You can edit this field and specify a different name. This is useful for sorting the services in reports (read-only).
Description	An editable field into which you can type information about the service.
Last update from original	The last date and time the WSDL was updated (read-only).
Original Location	The original location from where the WSDL was imported (read-only).
Override address	Enables you to enter an alternate endpoint for the service in the Service Address box.
Service Address	The endpoint of the service to which service requests are sent, retrieved from the WSDL file (read only). To override the default address, select Override address .
Service Name	The native service name in the WSDL file that is displayed by default when importing the service (read-only).

UI Elements (A-Z)	Description
Toolkit	The toolkit associated with the service. You set this when you import the service (read-only).
Update when opening script	Updates the WSDL from its source each time you open the script.

Connection Settings Tab

The following elements are included:

UI Elements (A-Z)	Description
Authentication	<p>Use Authentication Setting: Enables you to enter credentials for authentication.</p> <ul style="list-style-type: none"> ➤ Username, Password. the user name and password to use for retrieving the WSDL. <p>Tip: For users not in the default domain, type the domain name before the user name. For example, domain1/alex_qc.</p>
Proxy	<p>Use Proxy Setting. Enables you to enter proxy details and credentials.</p> <ul style="list-style-type: none"> ➤ Server. Name or IP address of proxy server. ➤ Port. Port through which to access the WSDL. ➤ Username, Password. the user name and password to be used for authentication. For users not in the default domain, type the domain name before the user name. For example, domain1/alex_qc.

UDDI Data Tab

The following elements are included:

UI Elements (A-Z)	Description
Service Key	A unique identifier of the service on the UDDI server, used to locate the service definition when updating the service.
UDDI Server	The URL address and version of the UDDI server from which the service definition is imported.
UDDI Version	The version of the UDDI registry: 2 or 3 .

Connection Settings Dialog Box

Enables you to provide authentication credentials and proxy server details for the machine hosting the WSDL file.

To access	For a new service: Select SOA Tools > Import Service . In the Import Services dialog box, click Connection Settings . For existing services: Select a service in the Service Management dialog box, and click the Connection tab.
Important information	Only available for services imported through a URL and UDDI.
Relevant tasks	"Import a service" on page 991

The following elements are included:

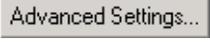
UI Elements (A-Z)	Description
Authentication	<p>Use Authentication Setting: Enables you to enter credentials for authentication.</p> <ul style="list-style-type: none"> ▶ Username, Password. the user name and password to use for retrieving the WSDL. <p>Tip: For users not in the default domain, type the domain name before the user name. For example, domain1/alex_qc.</p>
Proxy	<p>Use Proxy Setting. Enables you to enter proxy details and credentials.</p> <ul style="list-style-type: none"> ▶ Server. Name or IP address of proxy server. ▶ Port. Port through which to access the WSDL. ▶ Username, Password. the user name and password to be used for authentication. For users not in the default domain, type the domain name before the user name. For example, domain1/alex_qc.

Import Service Dialog Box

Enables you to import WSDLs from a file system, a URL, Application Lifecycle Management, a UDDI, or Systinet.

To access	<p>Use one of the following:</p> <ul style="list-style-type: none"> ▶ Select Services > New > Import Services ▶ Select New > Import Services from the shortcut menu
Relevant tasks	"Import a service" on page 991

The following elements are included (unlabeled GUI elements are shown in angle brackets>):

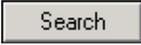
UI Elements (A-Z)	Description
	Browse. Enables you to locate a service on the file system, through a browser, UDDI registry, or Application Lifecycle Management repository depending on your Import WSDL from selection.
	Opens the Connections Settings dialog box for configuring the authentication and proxy settings of the server hosting the WSDL. For details, see "Connection Settings Dialog Box" on page 1007.
	Allows you to select a toolkit for the test. Choose Automatic , .NET , or Axis . The Automatic setting uses an algorithm to determine the most suitable toolkit.
	Opens the Application Lifecycle Management Connection dialog box to allow you to specify the ALM host machine.
	Begins the import process.
Select WSDL from	Location of WSDL. Browse for the information or enter it manually: <ul style="list-style-type: none"> ▶ File: Full path and file name ▶ URL: Complete URL. Make sure to insert a complete URL—not a shortened version ▶ ALM: Folder name in Application Lifecycle Management. If you are connected to a project, the Browse button opens the Import Service from Application Lifecycle Management dialog box. ▶ UDDI: UDDI registry ID. The Browse button opens the "Search for Service in UDDI Dialog Box" on page 1010.

Search for Service in UDDI Dialog Box

This dialog box enables you to locate a specific service from a UDDI registry.

To access	<ul style="list-style-type: none"> ▶ In the Manage Services window, click Import. ▶ In the Import dialog box, select UDDI in the Select WSDL from section. ▶ Click .
Relevant tasks	"Import a service" on page 991

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

UI Elements (A-Z)	Description
	Begins the search for a service based on the text in the All or part of the service name box.
<service list>	An alphabetical list of all the services that match the string. The grid shows the following columns: Service Name , Service Key , Service Description , Service WSDL .
All or part of the service name	<p>A string including the desired service name or part of the name. You do not need to use wildcard expressions.</p> <p>The following options narrow the search:</p> <ul style="list-style-type: none"> ▶ Exact Match. The service name must exactly match your text. ▶ Case Sensitive. The case of service name must match the case of the specified text.
UDDI server inquiry address	The complete path for the inquiry on the UDDI server. For example,
UDDI Version 2/3	The UDDI version of the services to display in the list.

XML/WSDL Comparison Dialog Box

This dialog box enables you to configure the settings for comparing different versions of a WSDL. You can instruct the comparison tool to ignore specific differences such as case, comments, and so forth.

To access	SOA Tools > SOA Settings > XML/WSDL Comparison.
Relevant tasks	"Check for WSDL updates - optional" on page 992.

User interface elements are described below:

UI Elements	Description
Show only differences	Show only differences in the report—do not display the matching text.
Ignore case	Do not show case mismatches as differences.
Ignore comments	Do not mark mismatches in the comment as differences.
Ignore processing instructions	Do not mark mismatches in the processing instructions as differences.
Ignore namespaces	Do not mark mismatches in namespaces as differences.

SOA Tools User Interface

This section includes (in alphabetical order):

- Manage Services Dialog Box on page 1003
- Connection Settings Dialog Box on page 1007
- Import Service Dialog Box on page 1008
- Search for Service in UDDI Dialog Box on page 1010
- XML/WSDL Comparison Dialog Box on page 1011
- Validate XML Dialog Box on page 1012
- WSDL Reference Analyzer Dialog Box on page 1014

- ▶ XML Editor Dialog Box on page 1015
- ▶ Find XML Dialog Box on page 1016

Validate XML Dialog Box

This dialog box enables you to validate XML parameters and their expected values.

To access	 on the upper toolbar.
Important information	Only available with a Service Test license.
Relevant tasks	"How to Validate the XML" on page 994

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

UI Elements	Description
	Reloads the XSD schema file in the checkpoint grid.
	Selects the Validate box for all elements.
	Clears the Validate box for all elements.
	<ul style="list-style-type: none"> ▶ In the upper pane: Deletes all checkpoint expected values. ▶ In the lower pane: Deletes all advanced checkpoints.
	Loads expected values for the XML elements from the recorded data.
	Loads expected values for the XML elements from the latest replay data.
	Deletes the selected advanced checkpoint.

UI Elements	Description
<checkpoints grid>	<p>A list of the response elements. The grid columns include:</p> <p>XML Checkpoints</p> <ul style="list-style-type: none"> ▶ The response elements you want to check <p>Validate</p> <ul style="list-style-type: none"> ▶ Enables/disables the validation for a specific element. <p>Expected Value</p> <ul style="list-style-type: none"> ▶  Comparison operator dropdown displaying the relevant comparison method such as Equals, Does not equal. ▶ The value to compare to the response. ▶  Enables the using of a parameter for the expected value instead of a hard-coded value.
<advanced checkpoints grid>	<p>A list of the response elements. The grid columns include:</p> <ul style="list-style-type: none"> ▶ XPath Query. The XPATH expression to use in the comparison. ▶ Validation Method. The validation method: Contains, Regular Expression, or Exact Phrase. ▶ Expected Value. The value to compare to the response.
XML Source	<p>The XML parameter to validate. The dropdown provides a list of all available parameters and the option to create a new one.</p> <ul style="list-style-type: none"> ▶ Show only output parameters. Limits the XML source dropdown list to show only output parameters.
Check for well-formed XML	Checks for well-formed XML, providing the results in the report.
XML Schema	The schema file associated with the XML. In some instances this is retrieved automatically.
Checkpoints	Enables checkpoint validation during the test run.

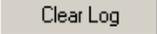
UI Elements	Description
Advanced Checkpoints	Enables the advanced checkpoint validation during the test run.
Fail test upon validation error	Marks the test status as Failed if a validation error occurs.

WSDL Reference Analyzer Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	SOA Tools > WSDL Reference Analyzer
Relevant tasks	"How to Analyze WSDL Dependencies" on page 993

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

UI Elements	Description
	Begins the analysis, showing all the results in the Log window.
	Clears the log window and log file.
	Opens the directory containing the output file.
<log window>	A running log of the reference analysis.
Select WSDL file	The local path or URL of the WSDL file to analyze.
Output file path	A location for the output zip file.

XML Editor Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	SOA Tools > XML Editor
Important information	Only available when installing a Service Test license.
Relevant tasks	"How to Edit the Schema" on page 996

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

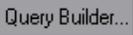
UI Elements	Description
	Load Schema. Opens an XSD schema file in the editor.
	Add Column. Adds a Values column to the right of the last column.
	Duplicate Column. Creates a new column with the same values as the selected column.
	Delete Column. Deletes the selected Values column.
	Load XML. Loads XML data from a file into the selected column.
	Edit XML. Opens the schema with the data of the selected column, in a text editor.
	Save XML. Saves the values from the selected column to an XML file.
<schema>	The XSD schema with the element values in grid format.
XML Values	An XML representation of the schema with the values of the selected column (right pane).

Find XML Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	Find XPATH in Tree view's Snapshot tab.
Important information	Only available with a Service Test license. This dialog box is used for both the Web Services and Flex protocols. For background information, see "XML Queries" on page 990.
Relevant tasks	"How to Build an XML Query" on page 1001 "How to Query an XML Tree" on page 662

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

UI Elements	Description
	Opens the XML Node Query dialog box for specifying query information.
XPath Query	The query expression. This can be entered manually or through the Query Builder.
Find	The SOAP message upon which to perform the search: Request or Response .

XML Node Query Dialog Box

User interface elements are described below:

UI Elements (A-Z)	Description
	Add Attribute. Opens the Attribute Properties dialog box.
	Delete. Removes the selected attribute from the list.
	Edit. Enables you to edit the selected attribute.
Name	Searches for the name of a node or element.

UI Elements (A-Z)	Description
Namespace URI	Searches for a namespace.
Text	Searches for the value of the element indicated in the Name box.
Attributes	Searches for an attribute in the attribute list.

37

Web Services - Preparing Scripts for Replay

This chapter includes:

Concepts

- ▶ Preparing for Replay Overview on page 1021
- ▶ Testing Web Service Transport Layers Overview on page 1025
- ▶ JMS Transport Overview on page 1026
- ▶ Asynchronous Messages Overview on page 1029
- ▶ Database Integration Overview on page 1032
- ▶ Negative Testing Overview on page 1040
- ▶ Customizing Overview on page 1042
- ▶ User Handler Examples on page 1046

Tasks

- ▶ How to Prepare Scripts for Replay on page 1051
- ▶ How to Set up Checkpoints on page 1054
- ▶ How to Send Messages over JMS on page 1055
- ▶ How to Send Messages over HTTP/S on page 1057
- ▶ How to Define a Testing Method on page 1059
- ▶ How to Add a Database Connection on page 1061
- ▶ How to Create a User Handler on page 1062
- ▶ How to Customize Configuration Files on page 1067

Reference

- ▶ Tree View Tabs on page 1068
- ▶ Database Integration User Interface on page 1071

Concepts

Preparing for Replay Overview

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

You can enhance it with custom error and log messages or with transactions. In addition, you can enhance your script with JMS functions, **jms_<suffix>** or XML functions, **lr_xml_<suffix>**. For more information, see the *Online Function Reference (Help > Function Reference)*.

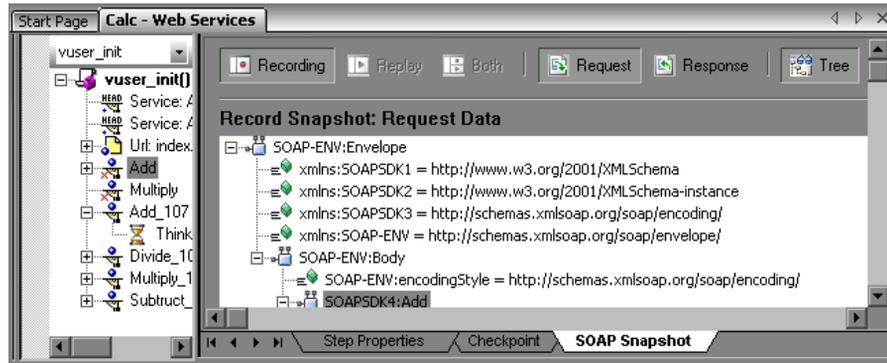
Run-time settings let you emulate real users more accurately, configure the run-time settings. These settings include general settings and Web Service specific settings. For details, see "Run-Time Settings" on page 417.

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 "Checkpoints" on page 1022. Checkpoint validation is only available with a Service Test license. For more information, contact HP support.

In certain cases, you may need to use the result of one Web Service call as input for another. To do this, you save the result to a parameter and reference it later. For more information, see "Assign input parameter values" on page 1051

Web Service Script Tabs

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 "New Web Service Call Dialog Box" on page 945.
- ▶ **Checkpoint.** A list of checkpoints defined for the step.
- ▶ **SOAP Snapshot.** A snapshot of the SOAP request and response for both record and replay. See "Snapshot Tab" on page 1068.

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.

TO set checkpoints, you set expected values for the arguments., Before replaying a test. You can enter values manually or load a set of expected values as they were captured either during recording or replay. This is useful when you have many argument values—instead of manually entering values, you automatically load them.

After the test run, you can view the Replay log or the test results and determine if the results were as expected.

The script generates a checkpoint argument for each row that you enable in the **Checkpoint** tab.

```
web_service_call("StepName=Add_2",
    "SOAPMethod=Calc.CalcSoapPort.Add",
    ...
    BEGIN_CHECKPOINTS,
        StopOnValidationError=1,
        CHECKPOINT, "XPATH=Result[1]", "Value=13",
        CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
    END_CHECKPOINTS,
    LAST);
```

The Checkpoint validation also provides support for standard XPATH validations using **lr_xml_find**.

For verification of the SOAP body (or SOAP headers with the .NET toolkit), you can use checkpoints. However, when using a toolkit other than .NET, checkpoints are not supported for SOAP headers—instead use XML Validation. For details, see "XML Validation Overview" on page 980.

For task details, see "How to Set up Checkpoints" on page 1054.

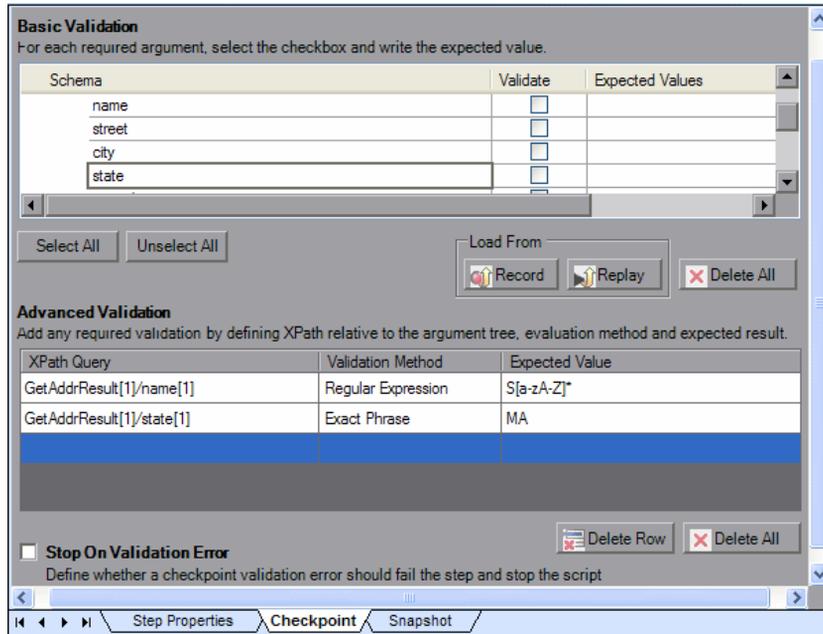
Basic and Advanced Validation

You can perform both **Basic** or **Advanced** validations.

In **Basic Validation**, VuGen with Service Test looks for exact matches of the value in the **Expected value** column. You can load expected values from a Recorded or Replay snapshot.

Use **Advanced Validation** to validate a checkpoint on non-leaf nodes or to define expected values in terms of a regular expression.

You can define both basic and advanced validations for the same step.



Note that when you select a non-leaf node, you need to supply all of the XML beneath the node.

In the Vuser script, VuGen indicates an exact match by **Value=** and a regular expression with **Expression=**:

```
BEGIN_CHECKPOINTS,
    CHECKPOINT, "XPATH=/AddResult[1]", "Value=50"
    CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
END_CHECKPOINTS,
```

XPath Expressions

Checkpoint expressions support both Nodeset and extended XPath syntax.

NodeSet expressions are XPath expressions evaluated to a single XML node or to a set of XML nodes. For example:

```
/a/b/c  
/x/y[1]  
//a/b
```

VuGen also supports full XPath expressions, such as **count(/a/b/c)** which is evaluated to int, or **count(/a/b/c) < 3** which is evaluated to a boolean value. The XPath support allows you to do numeric comparisons. For example, to verify whether a result is between 8 and 16, you can use the following expression:

```
number(/a/b/c) > 8 and number(/a/b/c) < 16
```

Testing Web Service Transport Layers Overview

Web services can be sent over various transport layers. The transport layer is the protocol used to transport messages to and from the server.

VuGen allows you to configure the transport layer for your services. It fully supports HTTP/HTTPS and JMS (Java Message Service) transport layers.

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 "User Handlers" on page 1042.

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.

Service Test lets you emulate asynchronous messaging for your Web Service call. For details, see "Sending Asynchronous Calls with HTTP/HTTPS" on page 1029.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls in conjunction with WS-Addressing. For details, see "WS-Addressing" on page 1030.

JMS Transport Overview

JMS is a J2EE standard for sending messages, either text or Java objects, between Java clients.

There are two scenarios for communication:

Peer-to-Peer. Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.

Publish-Subscribe. Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

VuGen supports point-to-point communication by allowing you to send and receive JMS messages to and from a queue.

Before you can send messages over JMS transport, you need to configure several items that describe the transport:

- ▶ **JNDI initial context factory.** The class name of the factory class that creates an initial context which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- ▶ **JNDI provider.** The URL of the service provider which will be used to locate the JMS resources such as JMS connection factory or JMS queue.

- **JMS connection factory.** The JNDI name of the JMS connection factory.

In addition, you can set a timeout for received messages and the number of JMS connections per process.

You configure these settings through the JMS run-time settings. For details, see "JMS Advanced Node" on page 474.

This section also includes:

- "JMS Script Functions" on page 1027
- "For details about the JMS functions, see the Online Function Reference (Help > Function Reference or click F1 on the function)JMS Message Structure" on page 1028

JMS Script Functions

VuGen uses its API functions to implement the JMS transport. Each function begins with a **jms** prefix:

Function Name	Description
jms_publish_message_topic	Publishes messages to a specific topic
jms_receive_message_queue	Receives a message from a queue
jms_receive_message_topic	Receives published messages to a specific topic on a subscription.
jms_send_message_queue	Sends a message to a queue.
jms_send_receive_message_queue	Sends a message to a specified queue and receives a message from a specified queue.
jms_subscribe_topic	Creates a subscription for a topic.

jms_set_general_property	Sets a general property in the user context.
jms_set_message_property	Sets a JMS header or property for the next message to be sent, or uses a JMS header or property to filter received messages.

The JMS steps/functions are only available when manually creating scripts—you cannot record JMS messages sent between the client and server.

Unlike peer-to-peer communication that uses message queues, the publish-subscribe functions, **jms_publish_message_topic**, **jms_subscribe_topic**, and **jms_receive_message_topic**, are not supported for Web Service calls. To use these functions with Web Service calls, you must manually set up user handlers to generate the JMS message payload. For more information, see "How to Create a User Handler" on page 1062.

For details about the JMS functions, see the *Online Function Reference* (**Help > Function Reference** or click **F1** on the function)  **JMS Message Structure**

Each JMS message is composed of:

- ▶ **Header.** contains standard attributes (Correlation ID, Priority, Expiration date).
- ▶ **Properties.** custom attributes.
- ▶ **Body.** text or binary information.

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

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*.

Asynchronous Messages Overview

You can use VuGen to emulate both synchronous and asynchronous messaging.

In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

This section also includes:

- "Sending Asynchronous Calls with HTTP/HTTPS" on page 1029
- "WS-Addressing" on page 1030

Sending Asynchronous Calls with HTTP/HTTPS

This following section describes how to use asynchronous calls in HTTP/HTTPS. You use a **Wait for Event** step to instruct Vusers to wait for the response of previous asynchronous requests before continuing. The listener blocks the execution of the service until the server responds.

When adding a Web Service Wait for Event step, you specify the following:

- **Quantifier.** The quantifier indicates whether the Vuser should wait for **ALL** events to receive a response or **ANY**, just one of them. **ANY** returns the name of the first event to receive a response. **ALL** returns one of the event names.
- **Timeout.** the timeout in milliseconds. If no events receive responses in the specified timeout, then **web_service_wait_for_event** returns a NULL.
- **Events.** a list all of the asynchronous events for which you want to wait.

When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

For task details, see "Send asynchronous HTTP messages - optional" on page 1057.

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 1030.

WS-Addressing

WS-Addressing is a specification that allows Web Services to communicate addressing information. It does this by identifying Web service endpoints in order to secure end-to-end endpoint identification in messages. This allows you to transmit messages through networks that have additional processing nodes such as endpoint managers, firewalls, and gateways. WS-Addressing supports Web Services messages traveling over both synchronous or asynchronous transports.

The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

An optional **WSAAction** argument allows you to define a SOAP action for instances where transport layers fails to send a message.

The following example illustrates a typical SOAP message using WS-Addressing, implemented in the background by VuGen.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2004/12/addressing">
  <S:Header>
    <wsa:MessageID>
      http://example.com/SomeUniqueMessageIdString
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://myClient.example/someClientUser</wsa:Address>
    </wsa:ReplyTo>
  <wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
  </wsa:FaultTo>
  <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
  <wsa:Action>http://myserver.example/DoAction</wsa:Action>
  </S:Header>
  <S:Body>
    <!-- Body of SOAP request message -->
  </S:Body>
</S:Envelope>
```

In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
web_service_call("StepName=Add_101",
  "SOAPMethod=Calc.CalcSoap.Add",
  "ResponseParam=response",
  "AsyncEvent=Event_1",
  "WSAReplyTo=212.199.95.138",
  "WSDL=http://lab1/WebServices/CalcWS/Calc.asmx?wsdl",
  "UseWSDLCopy=1",
  "Snapshot=t1153825715.inf",
  BEGIN_ARGUMENTS,
  "first=1",
  "second=2",
  END_ARGUMENTS,
  BEGIN_RESULT,
  "AddResult=Param_AddResult1",
  END_RESULT,
  LAST);
```

You can issue WS-Addressing calls in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, leave the **Async Event** box empty in the Transport Layer options. In Script view, remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

For task details, see "Send an asynchronous message using WS-Addressing - optional" on page 1058.

Database Integration Overview

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

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

Service Test saves all of the database interaction information and displays it in the Test Results report. For details, see Chapter 6, "Viewing Test Results."

The database integration functions are useful in the following scenarios:

- ▶ "Sending Messages over HTTP/HTTPS" on page 1025
- ▶ "Using Data Retrieved from SQL Queries" on page 1033
- ▶ "Validating Database Values after a Web Service Call" on page 1036
- ▶ "Checking Returned Values Through a Database" on page 1038
- ▶ "Performing Actions on Datasets" on page 1040

Connecting to a Database

To connect to a database, you add a connection step to your script. A built-in Connection String Generator guides you in creating a database connection string specific to your database and credentials. You can also test your connection before inserting the step.

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

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

For task details, see "How to Send Messages over JMS" on page 1055.

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 customer 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**. For information on the XML Validation tool, see "XML Validation Overview" on page 980.

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**).

Negative Testing Overview

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.

VuGen can check for a standard SOAP result or common SOAP fault responses. For example, if your Web Service attempts to access a Web page that cannot be found, resulting in a 404 HTTP error, using Negative testing the replay will fail, even if the SOAP is valid.

Testing Methods

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 VuGen to 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 VuGen to 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 Application Lifecycle Management or HP Service Test Manager, the application will list the test's status based on the Expected Response setting.

Customizing Overview

VuGen provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are user handlers and configuration files.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see below.

Configuration files let you customize advanced settings such as security information and the WSE configuration.

This section includes:

- ▶ "User Handlers" on page 1042
- ▶ "Custom Configuration Files" on page 1046

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 implement it through a DLL. You can apply the handler locally or globally. For details, see the following sections:

- ❑ Handler Function Definitions
- ❑ Event Handler Return Codes

For task details, see "How to Create a User Handler" on page 1062.

For sample user handlers, see "User Handler Examples" on page 1046.

Handler Function Definitions

For basic implementation of a user handler, you define a user handler function within your Vuser script with the following syntax:

```
int MyScriptFunction(const char* pArgs, int isRequest)
```

The **pArgs** argument contains the string that is specified in **UserHandlerArgs** argument of **web_service_call** function.

The **isRequest** argument indicates whether the function is being called during processing of a Request (1) or Response (0) SOAP envelope.

The content of SOAP envelope is passed to a parameter called **SoapEnvelopeParam** for both requests and responses. After the function processes the SOAP envelope, make sure to store it in the same parameter.

To call the handler function, use the function name as a value for the **UserHandlerFunction** argument in the relevant Web Service Call step. For more information, see the *Online Function Reference (Help > Function Reference)*.

Event Handler Return Codes

VuGen recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_SUCCEEDED_AND_MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in SoapEnvelopeParam .

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}

Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    return 0;
}
```

Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration. These files let you control the behavior of the test during run time.

The standard .NET configuration file, **mmdrv.exe.config**, is located in the VuGen installation folder. Some applications have their own configuration file, **app.config**.

You can customize the test run further, by filtering out the input or output. In addition, you can configure security information, such as token information and whether or not to allow unsigned test certificates.

For task details, see "How to Customize Configuration Files" on page 1067.

User Handler Examples

This section illustrates several common uses for user handlers.

.NET Filters

You can apply a .NET filter to your messages using the user handler mechanism.

If you are familiar with Microsoft's Web Service Enhancements (WSE) 2.0, you can create a .NET filter and register it for incoming or outgoing SOAP messages. A .NET filter is a class that is derived from `Microsoft.Web.Services2.SoapInputFilter` or `Microsoft.Web.Services2.SoapOutputFilter`. By overriding the **ProcessMessage** function of this class, you can examine and modify the envelope's body and header.

To define the filter globally for the entire script, add the following lines to the script's default.cfg file below.

```
[UserHandler]
Function=LrWsSoapFilterLoader
Args=<Filters InputFilterClass="class name" InputFilterLib="lib name"
OutputFilterClass="class name" OutputFilterLib="lib name" />
Order=BeforeSecurity/AfterSecurity/AfterAttachments
```

The **InputFilterClass** parameter indicates the name of your class, and **InputFilterLib** indicates the name of the assembly in which the class resides. For example:

```
web_service_call(
...
  "UserHandlerName=LrWsSoapFilterLoader",
  "UserHandlerArgs=<Filters
InputFilterClass="MyFilterNamespace.MyFilterClassName"
InputFilterLib="MyAssemblyName" />",
  BEGIN_ARGUMENTS,
  ...
  END_ARGUMENTS,
  ...
);
```

Use **SoapOutputFilter** to examine an outgoing **web_service_call** request, and **SoapInputFilter** to examine the response from the server. Use **InputFilterClass** and **InputFilterLib** if your filter is derived from **SoapInputFilter**, or **OutputFilterClass** and **OutputFilterLib** if your filter is derived from **SoapOutputFilter**.

To define the filter for a specific step, add the following arguments to the **web_service_call** function.

```
UserHandlerName= LrWsSoapFilterLoader
UserHandlerArgs=<Filters InputFilterClass="class name" InputFilterLib="lib name"
OutputFilterClass="class name" OutputFilterLib="lib name" />
UserHandlerOrder=BeforeSecurity/AfterSecurity/AfterAttachments
```

Overriding the Transport Layer

The following example shows a user handler function overriding the transport layer. VuGen does not automatically send the SOAP request over HTTP transport—instead it follows the transport method indicated in the custom handler.

After you receive a response, set the response envelope with the command:

```
lr_save_string(someResponseEnvelopeStr, "SoapEnvelopeParam");
```

To apply an alternate transport layer, specify **ReplaceTransport** as a value for the **UserHandlerOrder** argument. Define the transport layer in the handler.

```
web_service_call(  
...  
"UserHandlerFunction=<Transport HandlerFunction>",  
"UserHandlerArgs=<handler arguments>",  
"UserHandlerOrder=ReplaceTransport"  
...  
LAST);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the `web_service_call`:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
  "UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
  "<cls>"
  "<i>123456789</i>"
  "<s>abcde</s>"
  "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
  LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the actual path and content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the `web_service_call`:

```
"UserHandlerName=LrWsAttachmentsHandler",
"UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same `web_service_call`, modify the Web Service call as shown below:

```
"UserHandlerName=LrWsAttachmentsHandler",  
"UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;  
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;  
FileName=C:\\temp\\results.discomap",
```

Tasks

How to Prepare Scripts for Replay

This task describes how to prepare the script for replay and run it.

This task includes the following steps:

- "Assign input parameter values" on page 1051
- "Set the Run-time setting - optional" on page 1052
- "Configure XSDs with Any type elements - optional" on page 1052
- "Run the script" on page 1053
- "Review the test results" on page 1053

Assign input parameter values

First save the output result to a parameter, and then reference that parameter in a later Web Service call.

1 Save the output parameter.

- a** In Tree view, double-click the Web Service call whose output you want to use, to view its properties.
- b** In the left pane, select the output argument whose value you want to save to a parameter.
- c** In the right pane, select **Save returned value in parameter**. Specify a name in the **Parameter** box.

2 Use the saved parameter for input.

- a** In Tree view, double-click the Web Service call whose input parameters you want to set.
- b** In the left pane, select the input argument for which to use the saved parameter.
- c** In the right pane, select **Value**, and click on the ABC icon. The Select or Create Parameter box opens.

- d Select the saved output parameter from the **Parameter name** list.
- e To specify an input parameter in Script view, select the value you want to replace and select **Use Existing Parameters** from the shortcut menu. Select one of the available parameters.

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.

Set the Run-time setting - optional

Open the run-time settings (F4) to configure JMS and VM settings. Click the **JMS > Advanced** node. For user interface details, see the "JMS Advanced Node" on page 474.

Configure XSDs with Any type elements - optional

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

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

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

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

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

Set up Checkpoints

Set up checkpoints or XML validation. For details, see "How to Set up Checkpoints" on page 1054 or "How to Validate the XML" on page 994.

Tip: XML validation is recommended as it offers a more comprehensive validation of the response. It lets you insert independent validation steps into your script and load a complete XML tree as an expected value.

Run the script

Click **Vuser > Run**. Observe the output log for relevant messages.

Review the test results

The Test Results viewer opens automatically after the test run. An X indicates a failed step.

Expand the nodes to see detailed information about the SOAP response and checkpoints. For details, see "Viewing Test Results" on page 165.

How to Set up Checkpoints

This task describes how to validate the XML to be well-formed and verify the expected responses. For an overview about checkpoints, see "Checkpoints" on page 1022.

This task includes the following steps:

- "Open the Checkpoint tab" on page 1054
- "Set expected values" on page 1054
- "Enable the checkpoints" on page 1054
- "Set advanced checkpoints - optional" on page 1054
- "Run the script and view the results" on page 1055

1 Open the Checkpoint tab

In Tree view, select a step and then click the **Checkpoint** tab. For user interface details, see "Checkpoint Tab" on page 1070.

2 Set expected values

Specify a value in the **Expected Value** column or click the **Record** or **Replay** buttons to load values from the SOAP message.

3 Enable the checkpoints

In the **Validate** column, check the results that you want to validate. Click **Select All** to enable validation for all of the results.

4 Set advanced checkpoints - optional

- a** Select the **Advanced Checkpoints** option.
- b** In the **Advanced Validation** section, copy an XPath expression from an existing argument. For example, in the Checkpoint list, select an argument value and select Copy XPATH from the shortcut menu.
- c** Paste the XPath expression into the **XPath Query** column. For more information, see "XPath Expressions" on page 982.
- d** Select a Validation Method: **Regular Expression** or **Exact Phrase**.

- e Specify an expected value.

5 Run the script and view the results

- a Click F5 to run the script. View the Replay log to check for errors.
- b View the test results. If the Test Results window did not open automatically, select **View > Test Results**.
- c Expand the Checkpoint node. View the actual and expected values.

How to Send Messages over JMS

This task describes how to send messages using the JMS transport method.

This task includes the following steps:

- "Prerequisite - Create a Web Service call" on page 1063
- "Assign input parameter values" on page 1051
- "Set the Run-time setting - optional" on page 1052
- "Send synchronous JMS messages - optional" on page 1056
- "Send asynchronous JMS messages - optional" on page 1056
- "Send messages over JMS using SOAP messages - optional" on page 1056

1 Open the step properties

In Tree view, select the step whose transport you want to set. Select **Properties** from the shortcut menu.

2 Select the JMS transport method

Select the **Transport Layer Configuration** node and choose **JMS Transport**.

For UI details, see "Transport Layer Configuration Node" on page 947.

3 Set the run-time settings - optional

Configure the run-time settings, For details, see "JMS Advanced Node" on page 474.

4 Send synchronous JMS messages - optional

Once you create a Web Service call and designate the transport method as JMS, VuGen sends the JMS messages in a synchronous manner. If desired, specify the queue information.

5 Send asynchronous JMS messages - optional

To implement asynchronous messages over JMS, you send the request or retrieve the response using JMS steps—not Web Service calls.

- a Click within the script at the desired location. Select **Insert > New Step** and expand the **JMS Functions** node.
- b Select a JMS function: **JMS Send Message Queue** sends a message to a queue. **JMS Receive Message Queue** receives a message from the queue.
- c Click **OK** to open the JMS function properties.
- d Specify a queue name and click **OK** to generate the JMS functions.

For additional information about these functions, see the *Online Function Reference* (**Help > Function Reference** or click **F1** on the function).

6 Send messages over JMS using SOAP messages - optional

To send messages over JMS, using the SOAP message and without a Web Service call:

- a Record SOAP messages using a standard Web protocol.
- b Click within the script at the desired location. Select **Insert > New Step** and expand the **JMS Functions** node.
- c Select a JMS function: **Send Message Queue** or **JMS Receive Message Queue**.
- d Click **OK** to open the JMS function properties.
- e Specify a queue name and click **OK** to generate the JMS functions.

For details, see the *Online Function Reference* (**Help > Function Reference** or click **F1** on the function).

How to Send Messages over HTTP/S

This task describes how to send messages using the HTTP transport method.

This task includes the following steps:

- "Open the step properties" on page 1057
- "Select the HTTP/S transport method" on page 1057
- "Send a HTTP synchronous message - optional" on page 1057
- "Send asynchronous HTTP messages - optional" on page 1057
- "Send an asynchronous message using WS-Addressing - optional" on page 1058

1 Open the step properties

In Tree view, select the step whose transport you want to set. Select **Properties** from the shortcut menu.

2 Select the HTTP/S transport method

Select the **Transport Layer Configuration** node and choose **HTTP/S Transport**.

3 Send a HTTP synchronous message - optional

To send messages in synchronous mode over HTTP, create a standard Web Service call, and do not enable the **Async Support** option.

4 Send asynchronous HTTP messages - optional

- a** Choose **HTTP/S Transport** and select the **Async Support** option.
- b** Type an event name in the **Async Event** box.
- c** Click **OK** to generate the Web Service call.
- d** Add a **Wait for Event** step. Select **Insert > New Step** and choose **Web Service Wait for Event**.

- e Specify a step name, a quantifier, and a timeout. Click **Add** and insert the name of the event that you defined in the previous step.

In Script view, VuGen indicates asynchronous messaging with the added parameter, **AsyncEvent**.

```
web_service_call("StepName=EchoString_101",
  "SOAPMethod=EchoRpcEncoded.EchoSoap.EchoString",
  "ResponseParam=response1",
  "Service=ExtendedECHO_rpc_encoded",
  "AsyncEvent=Event_1",
  "Snapshot=t1157371707.inf",
  BEGIN_ARGUMENTS,
  "sec=7",
  "strString=mytext",
  END_ARGUMENTS,
  BEGIN_RESULT,
  "EchoStringResult=first_call",
  END_RESULT,
  LAST);
```

The **AsyncEvent** flag instructs the Vuser to wait for the response of previous asynchronous service requests.

5 Send an asynchronous message using WS-Addressing - optional

- a Select the **Async Support** option and provide an event name in the **Async Event** box. This can be an arbitrary name.
- b Select **WSA Support**. In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. Autodetect is useful when running the same script on several different machines. The server will reply to the specified location when the event occurs.
- c Click **OK** to save the settings.
- d Instruct the Vuser to wait for an event. Select **Insert > New Step** and add a **Web Service Wait For Event** step after the Web Service call step.
- e Specify a step name, quantifier, and timeout. To add an event name, click **Add**. The Web Service will wait for the specified event before responding.

- f Use the **Edit**, **Move Up**, and **Move Down** buttons to manipulate the events.

How to Define a Testing Method

This task describes how to select a testing method.

This task includes the following steps:

- "Prerequisite - Create a Web Service call" on page 1063
- "Assign input parameter values" on page 1051
- "Set the Run-time setting - optional" on page 1052
- "Verify function in the script" on page 1060
- "Run the script" on page 1053

1 Open the step properties

Select the step whose response you want to test. Open the **Properties** from the right-click menu.

2 Select an argument

Select the Output Argument node. For details, see "Output Arguments Node" on page 951.

3 Select a testing method and choose an expected response

- To perform negative testing only, select the **Negative Testing** check box and choose **SOAP Fault** as the **Expected Response**.

- ▶ To accept any type of SOAP response, select the **Negative Testing** check box and choose **Any SOAP** as the **Expected Response**.
- ▶ To perform positive testing only, clear the **Negative Testing** check box.

4 Verify function in the script

In Script view, VuGen indicates the testing method with the **ExpectedResponse** argument. In the following example, the script performs negative testing, indicated by the **SoapFault** value:

```
web_service_call("StepName=AddAddr_101",
  "SOAPMethod=AddrBook|AddrBookSoapPort|AddAddr",
  "ResponseParam=response",
  "Service=AddrBook",
  "ExpectedResponse=SoapFault",
  "Snapshot=t1189409011.inf",
  BEGIN_ARGUMENTS,
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
  LAST);
```

5 Evaluate the SOAP fault value

When you replay a script that results in a SOAP fault, VuGen saves the fault to a parameter called **response**. To check the returned value of the SOAP fault, evaluate the **response** output parameter using **lr_xml_find**.

In the following example, **lr_xml_find** checks for a **VersionMismatch** SOAP fault and issues an output message.

```
lr_xml_find("XML={response}",
  "FastQuery=/Envelope/Body/Fault/faultString ",
  "Value=VersionMismatch",
  LAST);

if (soap_fault_cnt > 0)
  lr_output_message("A Version Mismatch SOAP Fault occurred")
```

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

How to Add a Database Connection

This task describes how to add a database connection step through Tree view.

This task includes the following steps:

- "Prerequisite - Create a Web Service call" on page 1063
- "Assign input parameter values" on page 1051
- "Set the Run-time setting - optional" on page 1052
- "Verify function in the script" on page 1060
- "Run the script" on page 1053

1 Open Tree view

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

2 Select a section

Select the desired section: **vuser_init** or **Action**. To avoid repeating the connection sequence in every iteration, place it in the **vuser_init** section.

3 Insert a database connection step

Select **Insert > New Step**. Choose the **Database: Connect** step. The Database Connection dialog box opens. Specify a **Step Name**, **Connection Name**, and **Data Provider**, OLEDB or SQL.

4 Create a database connection string

- a** Click **Connection String Generator** to generate a database connection string specific to your environment.
- b** Indicate the connection properties:
 - **Server Name**
 - **Database Name**
 - **Authentication** method: Windows Authentication or User/password.
 - **Username** and **Password**

- c Click **Test Connection** to verify that the information you provided is correct.
- d Select an **SQL Provider**, OLEDB or SQL, and click **Generate**.

5 Verify function in the script

Check that an `lr_db_connect` function was written to the script

How to Create a User Handler

This task describes how to write a user handler for your script. For details, see the "Customizing Overview" on page 1042.

This task includes the following steps:

- "Prerequisite - Create a Web Service call" on page 1063
- "Assign input parameter values" on page 1051
- "Call the user handler function" on page 1063
- "Evaluate the handler function" on page 1064
- "Set the Run-time setting - optional" on page 1052
- "Run the script" on page 1053
- "Copy the user handler to all required machines" on page 1066
- "Implement the user handler - optional" on page 1066

1 Prerequisite - Create a Web Service call

Import a WSDL file and create a standard Web Service Call. For details, see "Adding New Web Service Calls" on page 911.

2 Define a user handler function

Define a user handler before the Web Service call:

```
int MyScriptFunction(const char* pArgs, int isRequest)
{
  ...
}
```

3 Call the user handler function

Call the handler function by specifying the function name as a value for the **UserHandlerFunction** argument. in the Web Service Call.

```
web_service_call(
...
"UserHandlerFunction=MyScriptFunction",
"UserHandlerArgs=<handler arguments>",
LAST);
```

4 Evaluate the handler function

Evaluate the handler's return code to determine if it succeeded. Use the return codes as described in "Event Handler Return Codes" on page 1044.

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
```

5 Create a DLL file - optional

To define a user handler through a DLL, locate the API header file, **LrWsHandlerAPI.h** in the product's **include** directory.

You can use a sample Visual Studio project located in the samples/ WebServices/SampleWsHandler directory as a template for creating a handler. The sample retrieves the request and response envelope and saves it to a parameter. To use this sample, open it in Visual Studio and modify it as required. If you do not need to save the request/response to a parameter, you can remove that section of the sample.

After editing the sample, save it and compile the DLL. When you compile the project, Visual Studio places the **<user_handler_name>.DLL** file in the **bin** folder. If you compile the project from another location, or if you want to copy the DLL from one machine to another, make sure to place it in the **bin** directory.

6 Configure the user handler - optional

Declare the DLL user handler globally or locally.

To apply the user handler globally to all requests in the script, add the following section to the **default.cfg** file in the script's directory.

```
[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>
```

7 Copy the user handler to all required machines

Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the product's `/bin` folder.

If you copy the script to another machine, it retains the handler information, since it is defined in script's folder.

8 Implement the user handler - optional

To implement a user handler, you use the entry functions `HandleRequest` or `HandleResponse`. Both functions have a single parameter, `context`, whose properties you can set in your handler. Use the Get functions to retrieve properties, and Set functions to pass information from the replay framework to the handlers or between the handlers.

- **GetEnvelope.** Gets the envelope content. For example, example:
`const char * pEnvelope = context->GetEnvelope();`
- **GetEnvelopeLength.** Gets the envelope length
- **SetEnvelope.** Sets the envelope content and length. For example:
`string str("MySoapEnvelope...");
context->SetEnvelope(str.c_str(), str.length());`
- **SetContentType.** Sets a new value for HTTP header content type
- **LogMessage.** Issues a message to the replay log
- **GetArguments.** Gets the configuration arguments defined for the current handler in order to pass it to the DLL
- **GetProperty.** Gets a custom property value
- **SetProperty.** Sets a custom property value

For more information, see the comments in the `LrWsHandlerAPI.h` file located in the product's **include** folder.

How to Customize Configuration Files

The following steps describe how to modify configuration files. For details, see "Custom Configuration Files" on page 1046.

- "Locate the configuration file" on page 1067
- "Save the application's configuration file" on page 1067
- "Set the security - optional" on page 1067

Locate the configuration file

Determine the location of the configuration file. The standard .NET configuration file, `mmdrv.exe.config` is located in the product's **bin** folder. Some applications have their own file, `app.config`.

Save the application's configuration file

If your application has its own `app.config` file:

- To apply the configuration information globally to all scripts, save the `app.config` file as `mmdrv.exe.config` in the **bin** folder, overwriting the existing file.
- To apply the configuration information locally, specifically for this script, copy the `app.config` file to the script's folder. This overrides the `mmdrv.exe.config` file, and remains associated with this script even when you copy it to other machines.

Set the security - optional

By default, VuGen allows unsigned certificates to facilitate testing. To disallow unsigned certificates, modify the `allowTestRoot` flag in the `<security>` section to **false**.

```
<security>
  <x509 storeLocation="currentuser" allowTestRoot="false">
```

Reference

Tree View Tabs

This sections describes the tabs visible in Tree view—**Snapshots** and **Checkpoints**. For details about the **Properties** tab, see "New Web Service Call Dialog Box" on page 945.

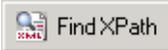
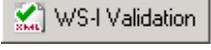
Snapshot Tab

This tab enables you to view a snapshot of the Web Service call. You can view both the record or replay snapshots, and the request and response.

To access	Snapshot tab in Tree view
Relevant tasks	"How to Prepare Scripts for Replay" on page 1051

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

UI Elements (A-Z)	Description
 Recording	Displays the recording snapshot.
 Replay	Displays the latest replay snapshot. Tip: To select a specific iteration, choose View > Snapshot > Choose Iteration .
 Both	Displays the latest recording and replay snapshots.
 Request	Displays the SOAP request sent to the server by the Web Service call.
 Response	Displays the SOAP response returned by the server.

UI Elements (A-Z)	Description
 Tree	Displays the SOAP message in a tree hierarchy. Tip: Expand the tree nodes to show the argument values.
 XML	Displays the SOAP message in XML textual format.
 Find XPath	Opens the Find XML dialog box and the Query Builder. For details, see "How to Build an XML Query" on page 1001. Note: Only available with a Service Test license.
 WS-I Validation	Checks WS-I compliance for the current snapshot. Note: Only available with a Service Test license.
<display area>	Shows the SOAP message based on your selections: <ul style="list-style-type: none"> ▶ Record, Response, or both snapshots ▶ Request or Response messages ▶ Tree or XML representation
shortcut menu	In the Response view, click on a output argument. <ul style="list-style-type: none"> ▶ Node Properties. Opens the XML Node Properties dialog box. ▶ Insert XML Check. Inserts an XML Find step. It allows you to specify a search string, case-sensitivity, and the behavior when errors occur. ▶ Save value in parameter. Saves the selected value to a simple parameter. ▶ Save XML in parameter. Saves the selected value to an XML parameter. ▶ Copy XML. Copies the XML node to the clipboard.

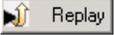
Checkpoint Tab

This dialog box enables you to validate XML parameters and their expected values.

To access	Checkpoint tab in Tree view
Important information	<ul style="list-style-type: none"> ▶ Only available with a Service Test license. ▶ For a more comprehensive validation of the response, use the XML Validation tool. This tool lets you insert independent validation steps into your script and load a complete XML tree as an expected value. For more information, see "How to Set up Checkpoints" on page 1054.
Relevant tasks	"How to Set up Checkpoints" on page 1054

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

UI Elements	Description
	Selects the Validate box for all elements.
	Clears the Validate box for all elements.
	<ul style="list-style-type: none"> ▶ In the Basic Validation pane: Deletes all expected values. ▶ In the Advanced Validation pane: Deletes all Advanced Validation entries.
	Deletes the selected Advanced Validation entry.
Basic Validation	<p>A list of the response elements. The columns include:</p> <ul style="list-style-type: none"> ▶ Schema. The response elements you want to check ▶ Validate. Enables/disables the validation for a specific element. ▶ Expected Value. The value to compare to the response.

UI Elements	Description
Load From	The source for the expected values: <ul style="list-style-type: none"> ▶ Record.  Takes the values from the recorded message ▶ Response.  Takes the values from the latest replay.
Advanced Validation	A list of the response elements. The grid columns include: <ul style="list-style-type: none"> ▶ XPath Query. The XPATH expression to use in the comparison. ▶ Validation Method. The validation method: Contains, Regular Expression, or Exact Phrase. ▶ Expected Value. The value to compare to the response.
Stop On Validation Error	Stops the test run and marks the test status as Failed if a validation error occurs.

Database Integration User Interface

This section includes (in alphabetical order):

- ▶ Snapshot Tab on page 1068
- ▶ Checkpoint Tab on page 1070
- ▶ Database Connection Dialog Box on page 1071
- ▶ Connection String Generator Dialog Box on page 1072

Database Connection Dialog Box

This dialog box helps you create a string to connect to your database.

To access	Click Connection String Generator in the Database Connection dialog box.
------------------	---

Relevant tasks	"How to Send Messages over JMS" on page 1055
See also	"Connection String Generator Dialog Box" on page 1072

User interface elements are described below:

UI Elements	Description
	Opens the Connection String Generator. For details, see "Connection String Generator Dialog Box" on page 1072.
Step Name	The name or IP address of the database server.
Connection String	The string by which to connect to the database. Use the Connection String Generator .
Data Provider	The SQL provider: OleDb or SQL .

Connection String Generator Dialog Box

This dialog box helps you create a string to connect to your database.

To access	Click Connection String Generator in the Database Connection dialog box.
Relevant tasks	"How to Send Messages over JMS" on page 1055
See also	"Database Connection Dialog Box" on page 1071

User interface elements are described below:

UI Elements	Description
	Tests the connection to the database.
	Generates the database connection string and writes it in the Connection String field in the Database Connection dialog box.
Server Name	The name or IP address of the database server.

UI Elements	Description
DB Name	The name of the database.
Authentication	The authentication method for the database: Windows Authentication or User/password . ► User Name, Password. The credentials for the database.
SQL Provider	The SQL provider: OLEDB or SQL .

38

Web Services - Security

This chapter includes:

Concepts

- ▶ Setting Security Overview on page 1076
- ▶ Security Scenarios Overview on page 1082
- ▶ WCF Scenario Settings on page 1087
- ▶ Advanced Scenario Setting on page 1092
- ▶ Preparing Security Scenarios for Running on page 1098

Tasks

- ▶ How to Add Security to a Web Service Script on page 1101
- ▶ How to Add SAML Security on page 1102
- ▶ How to Customize the Security on page 1104
- ▶ How to Create and Manage Security Scenarios on page 1109
- ▶ How to Parameterize Security Elements on page 1112

Reference

- ▶ Set Security User Interface on page 1114
- ▶ Security Scenario User Interface on page 1116

Web Services Security Examples on page 1119

Tips and Guidelines on page 1123

Concepts

Setting Security Overview

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

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

For more information on tokens, see below. For more information on SAML, see "SAML Security Options" on page 1081.

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** steps. For information on security scenarios, see "Security Scenarios Overview" on page 1082

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

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

Note: If your WSDL is located in a secure location, you must provide the security information through the Manage Services dialog box. For more information, see the "Connection Settings Dialog Box" on page 1007.

This section also includes:

- "Security Tokens and Encryption" on page 1077
- "SAML Security Options" on page 1081

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 Kerberos and Kerberos2 tokens is that Kerberos2 uses the Security Support Provider Interface (SSPI), so it does not require elevated privileges to impersonate the client's identity. In addition, the Kerberos2 security token can be used to secure SOAP messages sent to a Web Service running in a Web farm.

When you add a Kerberos token to the Vuser script, you specify a **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 details about the token attribute in the script, 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.

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

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

Working with Message Signatures and Encrypted Data

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

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

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

- ▶ **Digital Signatures.** Digital Signatures are used by message recipients to verify that messages were not altered since their signing. The digital signature is usually in the form of XML within the SOAP message. The recipient checks the signature to make sure it is valid. Certain environments, such as WSE, automatically verify the signature on the SOAP recipient's computer.
- ▶ **Encryption.** Although the XML digital signature offers a mechanism for verifying that the message has not been altered since it was signed, it does not encrypt the SOAP message—the message is still plain text in XML format. To secure the message in order that it should not be exposed, you encrypt it, making it difficult for an intruder to view and obtain a user's password.

Parameterization is not supported for message signatures and encryption arguments. For details on adding message signatures and encryption to your script, see "How to Add Security to a Web Service Script" on page 1101.

SAML Security Options

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

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

You can set the SAML settings for an entire script or part of the script. For details, see "How to Add SAML Security" on page 1102.

Note: You cannot apply SAML security and the standard Web Service (a **Web Service Set Security** step) security to the same step. To cancel Web Service security, insert a **Web Service Cancel Security** step.

Signing SAML Assertions

VuGen provides a method for signing an unsigned SAML assertion. As input, you provide the unsigned assertion, a certificate file, and the optional password. As output, VuGen provides the signed SAML assertion. For task details, see "How to Add SAML Security" on page 1102.

Policy Files

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

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

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

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

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

Security Scenarios Overview

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

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

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

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

For a "How To" guide on selecting a scenario, see "Tips and Guidelines" on page 1123.

This section includes:

- "Choosing a Security Model" on page 1083
- "Private, Imported, and Shared Scenarios" on page 1084
- "Scenario Categories" on page 1084

Choosing a Security Model

VuGen supports two models for configuring security for your Web Service calls: *Legacy* and *Scenario*. This chapter describes the Scenario security model. The Legacy model refers to the manual addition of Web Service Set Security steps, or the `web_service_set_security` function.

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

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

Private, Imported, and Shared Scenarios

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

You can select a scenario in three ways:

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

Scenario Categories

The scenario describes the configuration of your Web Service. It contains information such as security, encoding, proxy, and so forth. VuGen provides a Security Scenario editor that allows you to configure the settings for each scenario.

To determine the scenario that best fits your service, refer to the sections below. If you are unsure which scenario to choose, we recommend to use the **Custom Binding** scenario. For more information, see "The Custom Binding Scenarios" on page 1091.

Use the default **<no scenario>** for:

- ▶ simple Web Services where no advanced standards are required.
- ▶ scripts that use the legacy security model
- ▶ Web Services that require a specific security setting, not available in any of the existing scenarios.

If you select a built-in scenario and experience problems in replay, it is possible that no scenario was required and the problem is elsewhere. Reset the value to **<no scenario>**.

The built-in security scenarios are divided into the following categories:

- Core Scenarios
- Security Scenarios
- WCF Scenarios
- Optimization Scenarios

Core Scenarios

The following table describes the built-in Core scenario.

Scenario Name	When to use
Plain SOAP	<ul style="list-style-type: none"> ➤ Web services which do not require advanced standards ➤ Web services which may require you to specify the WS-Addressing version

For this type of scenario, if your service uses WS-Addressing, specify the version.

Security Scenarios

The following table describes the built-in Security scenario.

Scenario Name	When to use
Username Authentication	<ul style="list-style-type: none"> ➤ Client is authenticated with a username and password on the message level

For this type of scenario, specify the username/password, and if your service uses WS-Addressing, specify the version.

WCF Scenarios

The following table shows the scenarios for Web Services that utilize WCF. The WSHttpBinding-based scenarios are divided according to the way the client authenticates itself to the server. For example, if your client presents a user name and a password to the server, choose the **Username (message protection)** scenario. The user interface lets you provide the identity information in the form of a user name or a certificate as required.

WCF Scenario Name	When to use
WSHttpBinding - No Authentication	<ul style="list-style-type: none"> ▶ 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	<ul style="list-style-type: none"> ▶ 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	<ul style="list-style-type: none"> ▶ 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	<ul style="list-style-type: none"> ▶ 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	<ul style="list-style-type: none"> ▶ SSL is enabled ▶ Client is authenticated with a username and password ▶ Communication may utilize advanced standards such as secure conversation and MTOM

WCF Scenario Name	When to use
WSFederationHttpBinding	<ul style="list-style-type: none"> ➤ Client authenticates against the STS using a predefined scenario ➤ Client uses the token given from the STS to authenticate against the server
Custom Binding	<ul style="list-style-type: none"> ➤ Web Service that uses WS-* standards ➤ WCF services of any configuration

Optimization Scenarios

The following table describes the built-in Optimization scenario.

Scenario Name	When to use
MTOM	<ul style="list-style-type: none"> ➤ MTOM enabled Web services ➤ Web Services which may require you to specify the WS-Addressing version

For MTOM type scenarios, if your service uses WS-Addressing, specify the version.

WCF Scenario Settings

This section describes the values required for the WCF security scenarios:

This section includes:

- "The WsHttpBinding Scenario" on page 1088
- "The Federation Scenario" on page 1090
- "The Custom Binding Scenarios" on page 1091

The WsHttpBinding Scenario

No Authentication (Anonymous)

In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication.

You specify only one of the following settings:

- ▶ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- ▶ **Specify service certificate.** Browse for a service certificate. For more information, see "Select Certificate Dialog Box" on page 1118. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information.

- ▶ **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Windows Authentication

This WCF scenario uses Windows Authentication.

You declare the expected identity of the server in terms of its **SPN** or **UPN** identities. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.

Certificate Authentication

In this WCF WsHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Specify only one of the following settings:

- ▶ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- ▶ **Specify service certificate.** Browse for a service certificate. For details, see "Select Certificate Dialog Box" on page 1118. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username Authentication (Message Protection)

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For details, see "Select Certificate Dialog Box" on page 1118. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username (Transport Protection) Authentication

This WCF WSHttpBinding scenario enables SSL and authenticates the client with a user name and password on the message level.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

The Federation Scenario

In the **WSFederationHttpBinding** scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server.

Therefore, two bindings are needed, one against the STS and another against the application server.

First, use the Security Scenario editor to define an STS binding. For more information, see "Create a scenario (if you do not have existing ones)" on page 1109. When setting the binding against the application server, specify this file in the **Referenced file** box.

For the Federation scenario, specify the following server information:

- ▶ **Transport.** HTTP or HTTPS
- ▶ **Encoding.** Text or MTOM

For the Federation scenario, specify the following security information:

- ▶ **Authentication mode.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, IssuedTokenOverTransport, or SecureConversation
- ▶ **Bootstrap policy.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, or IssuedTokenOverTransport

For the Federation scenario, specify the following identity information:

- ▶ **Server certificate.** Browse for a server certificate. For more information, see the "Select Certificate Dialog Box" on page 1118.
- ▶ **Expected server DNS.** the expected identity of the server in terms of its DNS. This can be **localhost** or an IP address or server name.

For the Federation scenario, specify the following STS (Security Token Service) information:

- ▶ **Issuer address.** The address of the issuer of the STS. This can be **localhost**, an IP address, or a server name.

- **Referenced binding.** The file that references the binding that contacts the STS (Security Token Service)

The Custom Binding Scenarios

The **Custom Binding** scenario enables the highest degree of customization. Since it is based upon WCF **customBinding**, it allows you to test most WCF services, along with services on other platforms such as Java that use WS - *<spec_name>* specifications.

Use the **Custom Binding** scenario to configure a custom scenario that does not comply with any of the predefined security scenarios.

For the Custom Binding scenario, specify the following server information:

- **Transport.** HTTP, HTTPS, TCP, or NamedPipe
- **Encoding.** Text, MTOM, or WCF Binary

Specify the following security information:

- **Authentication mode.** None, AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SecureConversation, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Bootstrap policy.** For SecureConversation type authentication, specify a bootstrap policy: AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Net security.** the network security. Select None, Windows stream security, or SSL stream security. For services with HTTP transport, leave the default value, **None**. To enable SSL for HTTP, choose the HTTPS transport.

If your Web Service uses **Reliable messaging**, enable the option, and select **Ordered** or **Not Ordered**.

Identities

Your security settings may require you to provide identity details for either the client and server, or both of them.

An example of identity details for the client, are user name/password or an **X.509** certificate.

For identity information, provide one or more authentication details as required by the service:

Username, Password, Server certificate, Client certificate, or a custom Windows identity. For details about choosing a certificate, see "Select Certificate Dialog Box" on page 1118.

Some scenarios require you to declare the expected identity of the server in terms of its DNS, SPN, or UPN identity.

- **DNS.** Provide the name of a server or use localhost.
- **SPN.** Provide the SPN identity in the domain\machine format.
- **UPN.** Provide the UPN identity in the user@domain format.

After setting the basic values, you can set advanced attributes as described in "Advanced Scenario Setting" on page 1092

Advanced Scenario Setting

This section describes the Advanced scenario settings to customize a security scenario in the areas of Encoding, Advanced Standards, Security, or HTTP and Proxy.

Not all settings are relevant for all scenarios, so some of them might be disabled or hidden depending on the scenario.

This section includes:

- "Encoding" on page 1093
- "Advanced Standards" on page 1093
- "Security" on page 1094

- "HTTP and Proxy" on page 1096

Encoding

The Encoding tab lets you indicate the type of encoding to use for the messages: **Text**, **MTOM**, or **Binary**. The default is **Text** encoding.

For each of these encoding methods, you can choose a version of WS-Addressing:

- None
- WSA 1.0
- WSA 04/08

Advanced Standards

This tab lets you configure advanced WS- standards, such as Reliable Messaging and the Via address option.

If your service implements the **WS-ReliableMessaging** specification, enable the **Reliable Messaging** option and set the following options:

- **Reliable messaging ordered.** indicates whether the reliable session should be ordered
- **Reliable messaging version.** WSReliableMessagingFebruary2005 or WSReliableMessaging11

Via Address

In certain instances, you may need to send a message to an intermediate service that submits it to the actual server. This may also apply when you send the message to a debugging proxy. This corresponds to the WCF **clientVia** behavior.

In such cases it may be useful to separate the physical address to which the message is actually sent, from the logical address for which the message is intended. The logical address may be the physical address of the final server or any name. It appears in the SOAP message as follows:

```
<wsa:Action>http://myLogicalAddress<wsa:Action>
```

The logical address is retrieved from the user interface. By default, it is the address specified in the WSDL. You can override this address from the Manage Services dialog box.

Security

The Advanced security settings correspond to the **WS-Security** specifications.

For security scenarios that are based upon WCF WSHttpBinding, you can indicate the following settings:

- ▶ **Enable secure session.** Establish a security context using the WS-SecureConversation standard.
- ▶ **Negotiate service credentials.** Allow WCF proprietary negotiations to negotiate the service's security.

For **WSHttpBinding**, **Custom Binding**, or **WSFederationHttpBinding** WCF type scenarios, you can set the default algorithm suite and protection level:

Attribute	Meaning	Possible Values
Default Algorithm Suite	The algorithm to use for symmetric/asymmetric encryption. These are the values from the SecurityAlgorithmSuite configuration in WCF:	<ul style="list-style-type: none"> ▶ 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	<ul style="list-style-type: none"> ➤ 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	<ul style="list-style-type: none"> ➤ Strict ➤ Lax ➤ LaxTimeStampFirst ➤ LaxTimeStampLast
Key Entropy Mode	The entropy mode for the security key.	<ul style="list-style-type: none"> ➤ 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	<ul style="list-style-type: none"> ➤ Always to Recipient ➤ Never ➤ Once ➤ AlwaysToInitiator
X509 Reference Style	How to reference the certificate	<ul style="list-style-type: none"> ➤ Internal ➤ External
X509 require derived keys	Should X509 certificates require derived keys	<ul style="list-style-type: none"> ➤ Enable - Yes ➤ Disable - No
X509 key identifier clause type	The type of clause used to identify the X509 key.	<ul style="list-style-type: none"> ➤ Any ➤ Thumbprint ➤ IssuerSerial ➤ SubjectKeyIdentifier ➤ RawDataKeyIdentifier

HTTP and Proxy

This tab lets you set the HTTP and Proxy information for your test.

HTTP (S) Transport

The following table describes the HTTP(S) Transport options:

Option	Meaning	Possible Values
Transfer mode	The transfer method for requests/responses	Buffered, Streamed, StreamedRequest, StreamedResponse
Max response size (KB)	The maximum size of the response before being concatenated	Default 65 KB
Allow cookies	Enable cookies	Enabled/Disabled
Keep-Alive Enabled	Enable keep-alive connections	Enabled/Disabled

Option	Meaning	Possible Values
Authentication scheme	HTTP authentication method	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous
Realm	The realm of the authentication scheme	Any URL
Require client certificate	For SSL transport, require a certificate	Enabled/Disabled

Proxy Information

If the Web service's transport uses a proxy server, you can specify its details in the **Security** tab. The following table describes the proxy options:

Option	Meaning	Possible Values
Use default web proxy	Use machine's default proxy settings	Enabled/Disabled
Bypass proxy on local	Ignore proxy when the service is on the local machine	Enabled/Disabled
Proxy address	the proxy server	Any URL
Proxy authentication scheme	HTTP authentication method on Proxy	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

Preparing Security Scenarios for Running

The following sections describe several advanced modifications for your Web Services script that may be required for replay.

This section includes:

- ▶ "Parameterizing Security Elements" on page 1098
- ▶ "Protecting Custom Headers" on page 1098
- ▶ "Emulating Users with Iterations" on page 1099

Parameterizing Security Elements

You can parameterize the security elements in a script independently. For example, in a username-based security scenario, you might want each Vuser or iteration to use a different user name.

Protecting Custom Headers

When an operation uses SOAP headers, VuGen does not automatically sign or encrypt them. To incorporate a protection scheme such as a signature or encryption, you must manually add the following information to the scenario's configuration file (.stss) in the **behavior** element:

- ▶ soapAction of the relevant operation
- ▶ the header XML name and namespace
- ▶ the protection level

The following example shows an outgoing message with the soapAction, **http://mySoapAction**. The XML element **header1** from namespace **http://myServiceNamespace** is encrypted and signed. The **header2** element from the same namespace is only signed.

```

<protocols ...>
  ...
  <behaviors>
    <contractBehaviors>
      <behavior>
        <channelProtectionBehavior>
          <protectionRequirements action="http://mySoapAction">
            <incomingEncryptionParts>
              <header localName="header1" namespace="http://
myServiceNamespace" />
            </incomingEncryptionParts>
            <incomingSignatureParts>
              <header localName="header1" namespace=" http://
myServiceNamespace " />
            </incomingSignatureParts>
            <header localName="header2" namespace=" http://
myServiceNamespace " />
            </incomingSignatureParts>
          </protectionRequirements>
        </channelProtectionBehavior>
      </behavior>
    </contractBehaviors>
  </behaviors>
</protocols>

```

Emulating Users with Iterations

Many of the security scenarios establish a session with the server. For example, every scenario that uses **WS-SecureConversation** establishes a server session. This session is established when the first operation is executed and ends when the script is finished. By default, VuGen closes all sessions after each iteration and opens them again when the next iteration begins. This implies that every iteration simulates a new session and Vuser.

When working with multiple iterations, this may not be the desired effect—you may prefer to keep the original session active and not open a new session for each iteration. This applies when load testing through the LoadRunner Controller or when setting multiple iterations in the run-time settings.

You can override this behavior so that only the first iteration will establish a new session, while all subsequent ones will continue to use the open session. This simulates a user who repeatedly performs an action using the same session.

To determine which simulation mode to use, choose the one which is most appropriate to what you are simulating. For example, if you are simulating a load test where most of the actions are performed repeatedly by the same user in a single session, use the above configuration. If you are unsure, leave the default settings.

Tasks

How to Add Security to a Web Service Script

This task describes how to add set the security for your Web Service calls. For details about Web Services security, see "Setting Security Overview" on page 1076.

This task includes the following steps:

- "Insert a new Web Services Security Step" on page 1101
- "Add a token - optional" on page 1101
- "Add a message signature or encryption - optional" on page 1102
- "Set a message timeout - optional" on page 1102
- "Cancel the security settings - optional" on page 1102

Insert a new Web Services Security Step

- 1** Place the cursor at the point at which you want to add the security settings. In most cases, it is best to lace it in `vuser_init` so that the security scope will be applied to the whole script. To apply the security for specific calls, place it at the desired location.
- 2** Select **Insert > New Step** to open the Add Step dialog box.
- 3** Select **Web Service Set Security** and click **OK**. The Set Security Properties box opens.

Add a token - optional

- 1** Click **Add** to add a new token. The Add Token dialog box opens.
- 2** Select a token type. For details, see "Security Tokens and Encryption" on page 1077.

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**.

Add a message signature or encryption - optional

- 1** Click **Add > Message Signature** or **Add > Encrypted Data**.
- 2** Select a token to use with the message signature or encryption. Both signatures and encryptions require you to specify a token previously defined as the signing/encrypting token.
- 3** Specify a target token, or leave the field blank to apply the signature or encryption to the whole message body. For details, see "Security Tokens and Encryption" on page 1077.

See "Web Services Security Examples" on page 1119 to learn how to encrypt or sign a specific XPath in the SOAP.

Set a message timeout - optional

To specify a time for which the message packet is considered valid, select **Time To Live** and specify a time in seconds.

Cancel the security settings - optional

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

How to Add SAML Security

This task describes how to add SAML security for your Web Service calls. For more information about SAML security, see "SAML Security Options" on page 1081.

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

This task includes the following steps:

- "Insert a new Web Services Security step" on page 1103
- "Set the security policy - optional" on page 1104
- "Cancel the security settings - optional" on page 1102

1 Insert a new Web Services Security step

- a** Place the cursor at the point at which you want to add the security settings.
- b** Select **Insert > New Step** to open the Add Step dialog box.
- c** Select **Web Service Set Security SAML** and click **OK**. The properties box opens.

2 Insert a SAML assertion.

To add a SAML assertion method, add a **Web Service Sign SAML Assertion** step through the Add Step dialog box (**Insert > New Step**). Provide the unsigned assertion, a certificate file, and a password (optional).

3 Set the security policy - optional

Specify a policy file, or leave it blank to use the default. If you manually enter values, they override any values in the policy file. You must provide an Issuer URL, also known as the **STS URL**.

4 Cancel the SAML settings - optional

To remove the settings at a specific point in the script, insert a **Web Service Cancel Security SAML** step.

How to Customize the Security

This task describes how to how to configure special cases common to Web Service security.

This task includes the following steps:

- "Reference a token with a SubjectKeyIdentifier - optional" on page 1105
- "Customize the Username token - optional" on page 1106
- "Customize the encryption - optional" on page 1107
- "Customize WS-Security - optional" on page 1107

Reference a token with a SubjectKeyIdentifier - optional

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 SKI (Subject Key Identifier). This is common with tokens that are used for encryption.

- 1 Add a token as described in the "How to Add Security to a Web Service Script" on page 1101.
- 2 In the Add Token dialog box, set the Add option to **False**.

- 3 Alternatively, configure this setting in the script:

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

- 4 If necessary, set the **useRFC3280** settings as described in "Customize WS-Security - optional" on page 1107.

Customize the Username token - optional

You can customize the Username token with a nonce and timestamp.

- 1 Locate the `web_service_set_security` function in the script.
- 2 Add the attributes and their values according to this chart:

Name	Meaning	Possible values
IsNonceIncluded	Include a nonce with the token.	True (default) or False
TimestampFormat	The timestamp format to use with the token.	<ul style="list-style-type: none"> ➤ None. no timestamp ➤ Full. a <timestamp> element with <created> and <expired> inner elements ➤ Created. (default) only a <created> element

For example:

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

Customize the encryption - optional

You customize encryption by indicating whether to encrypt the whole element or only its content. This is common when encrypting tokens such as a user name. By default, only the content is encrypted.

To encrypt the entire token:

- 1** Locate the `web_service_set_security` function in the script.
- 2** Add the `EncryptionType` attribute with the value `Element`.

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

- 3** To return to the default, remove the `EncryptionType` attribute or set it to `Content`.

Customize WS-Security - optional

To change the algorithm Service Test uses for encryption or to modify some other low-level security details.

- 1** To change either of these items, open the `%Service Test%/bin/mmdrv.exe.config` file in a text editor.

- 2 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>
```

- 3 Set the element values as required:

Name	Meaning	Possible values
verifyTrusy	Check sent/received x.509 certificate's validity.	<ul style="list-style-type: none"> ▶ True (default) ▶ False
sessionKeyAlgorithm	The algorithm the session symmetric key should use to encrypt the message.	<ul style="list-style-type: none"> ▶ AES128 ▶ AES192 ▶ AES256 ▶ TripleDES
keyAlgorithm	The algorithm to use by the public key to encrypt the session key.	<ul style="list-style-type: none"> ▶ RSA15 ▶ RSAOAEP
useRFC3280	Generate subject key identifiers that are interoperable and not Windows specific.	<ul style="list-style-type: none"> ▶ True ▶ False (default)

How to Create and Manage Security Scenarios

The following steps describes how to create and customize a security scenario for a specific service.

- "Open the Security Scenario Data dialog box" on page 1109
- "Create a scenario (if you do not have existing ones)" on page 1109
- "Load a security scenario (if you have existing ones)" on page 1110
- "Configure advanced settings - optional" on page 1110
- "Modify an existing security scenario - optional" on page 1110

1 Open the Security Scenario Data dialog box

- a** Click **Manage Services**. In the left pane, select the service for which you want to set the security scenario. If necessary, import a service, as described in "Import Service Dialog Box" on page 1008.
- b** Select the **Protocol and Security** tab and click the **Edit Data** button. The Security Scenario Data dialog box opens.

2 Create a scenario (if you do not have existing ones)

- a** Choose **Private scenario** and select a built-on security scenario for the current service.
- b** In the **Scenario type** box, choose a scenario. For details, see "Choosing a Security Model" on page 1083.
- c** Specify the required values for your scenario. For details, see "WCF Scenario Settings" on page 1087.
- d** To specify a certificate (only applicable to some of the scenarios), click the Browse button adjacent to the **Client certificate** or **Specify service certificate** box to open the Select Certificate dialog box. For details, see the "Select Certificate Dialog Box" on page 1118.
 - To retrieve a certificate from a file, choose **File** and locate its path.

- ▶ To retrieve a certificate from a Windows store, Choose **Windows Store**. Select a Store location and name. Specify a search string—to search for all certificates, leave the **Search text** box empty. To search for a specific certificate, specify a substring of the certificate name. If required, specify a password for the private key. Click **Find** to generate the list of certificates found in the store.

3 Load a security scenario (if you have existing ones)

- a To use an existing scenario with the ability to modify it, choose **Private scenario**. Click **Import**. In the Shared Scenario dialog box, select a stored scenario. If required, modify the settings as described in "WCF Scenario Settings" on page 1087.
- b To use an existing scenario without the option of changing it, choose **Shared Scenario**. Use the Browse button to open the Shared Scenario dialog box and select a stored scenario.

Note: If someone modifies a shared scenario file at its source, it will affect your script.

4 Configure advanced settings - optional

Click **Advanced** to configure the Proxy, Encoding, and other advanced setting. For most scenarios, the default settings are ideal. For details, see "Advanced Scenario Setting" on page 1092. Click **OK** to save the security scenario.

5 Modify an existing security scenario - optional

To create and modify security scenarios that will be available globally for all scripts—not just this specific service, use the Security Scenario editor. You can also use the editor to save the scenario so that others may load it.

- a Choose **SOA Tools > Security Scenario Editor**.
- b Click the **Load** button and browse for an existing **stss** scenario file.
- c Modify the scenario settings as required

- d** Click **Save** or **Save as**.

6 Protect SOAP headers - optional

Manually modify the **behavior** element in the scenario's configuration file

- a** In VuGen, open the Script view. Choose **View > Script View**.
- b** Click in the script editor and select **Open Script Directory** from the shortcut menu.
- c** Locate the security scenario's configuration file `<service_name>.stss` in `WSDL/@config` directory.
- d** Modify the behavior section of the file. For details, see "Protecting Custom Headers" on page 1098.

7 Set the iteration mode- optional

To configure your environment to use the same session for all iterations:

- a** Open the script root directory: In Script view, click inside the script and choose **Open Script Directory** from the shortcut menu.
- b** Open `default.cfg` file in a text editor.
- c** In the `[WebServices]` section, add in a row under the toolkit. If you are using the Axis toolkit or if you configured other settings, the file contents may differ.

```
[WebServices]
Toolkit=.Net
SimulateNewUserInNewIteration=0
```

- d** Save and close the file.

For details, see "Emulating Users with Iterations" on page 1099

How to Parameterize Security Elements

This task describes how to independently parameterize the security elements in a script.

This task includes the following steps:

- "Open the Security Scenario Editor" on page 1113
- "Set up a scenario for each Vuser" on page 1113
- "Open the Parameter List window and create a parameter." on page 1113
- "Add parameter values" on page 1113
- "Call the parameter" on page 1113

1 Open the Security Scenario Editor

Select **SOA Tools > Security Scenario Editor**.

2 Set up a scenario for each Vuser

Set up a scenario for each Vuser as described in "How to Create and Manage Security Scenarios" on page 1109. We recommend you use the names **user1**, **user2**, and so forth, and save them in a new folder, **%script root%/WSDL/referencedConfig**.

3 Open the Parameter List window and create a parameter.

Select **Vuser > Parameters List**. Create a new parameter, **<ServiceName>_shared_config**. Replace the **<ServiceName>** with the case-sensitive name of the service you are testing. To determine the exact name of the service, click **Manage Services** to see the list of services.

4 Add parameter values

In the values table, in each row add the file names of the security scenarios with their **.stss** extensions. You can use a relative path, relative to the script directory. Click **Add Row** to add multiple values. Close the Parameter List dialog box.

5 Call the parameter

- a** Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.
- b** Select **Shared Scenario**. Click the **Browse** button and enter the parameter name, **<ServiceName>_shared_config**, in the test box.

Reference

Set Security User Interface

This section includes (in alphabetical order):

- ▶ Set Security Properties Dialog Box on page 1114
- ▶ Security Scenario Editor Dialog Box on page 1116
- ▶ Select Certificate Dialog Box on page 1118

Set Security Properties Dialog Box

This dialog box enables you to set the security properties for your Web Service calls.

To access	Insert > New Step > Web Service Set Security. Click OK.
Relevant tasks	"How to Add Security to a Web Service Script" on page 1101
See also	"How to Add SAML Security" on page 1102

User interface elements are described below:

UI Elements	Description
	Opens the Add Token or Add Message Signature, or Add Encrypted Data dialog boxes. The two latter options are only available after you define at least one token.
	Removes a security element from the list.
Time to Live (seconds)	The time for which the message packet is considered valid. Default. 60 seconds.

UI Elements	Description
Security Elements	The names and types of the security elements, such as tokens, message signatures, or data encryptions.
Attributes	<p>The attributes of the security elements. For a token, it includes the following:</p> <ul style="list-style-type: none"> ➤ User Name, Password. The credentials for the token. ➤ Password Options. How to send the password: SendPlainText etc. ➤ Add. Include the token in the header. <ul style="list-style-type: none"> ➤ True. Sends the token explicitly in the SOAP envelope header. ➤ False. Excludes the token from the SOAP envelope header.

Add Token Dialog Box

User interface elements are described below:

UI Elements	Description
Type	<p>A token type:</p> <ul style="list-style-type: none"> ➤ Username and Password ➤ X.509 Certificate ➤ Kerberos Ticket ➤ Kerberos2 Ticket ➤ Security Context Token ➤ Derived Token.

UI Elements	Description
Logical Name	An arbitrary name for the token used by VuGen to identify the token.
Properties	<p>The token attributes:</p> <ul style="list-style-type: none"> ▶ User Name, Password. The credentials for the token. ▶ Password Options. How to send the password: SendPlainText etc. ▶ Add. Include the token in the header. <ul style="list-style-type: none"> ▶ True. Sends the token explicitly in the SOAP envelope header. ▶ False. Excludes the token from the SOAP envelope header.

Security Scenario User Interface

This section includes:

- ▶ Set Security Properties Dialog Box on page 1114
- ▶ Security Scenario Editor Dialog Box on page 1116
- ▶ Select Certificate Dialog Box on page 1118

Security Scenario Editor Dialog Box

This dialog box enables you to define security scenarios for your script.

To access	SOA Tools > Security Scenario Editor
Important information	You can also define scenarios for a specific service. For details, see "How to Create and Manage Security Scenarios" on page 1109.
Relevant tasks	"Modify an existing security scenario - optional" on page 1110

User interface elements are described below:

UI Elements	Description
	New. Resets the editor for defining a new security scenario. If you made changes to the current scenario, it prompts you to save them.
	Load. Opens an existing shared scenario from a URL or file.
	Save. Saves the scenario file. If you have not saved the file at least once, it prompts you for a name.
 Save as	Save as. Saves the scenario file at a new location.
	Help. Opens the Online help for security scenarios.
	Close. Closes the dialog box.
	Opens the Advanced Setting dialog box for setting the encoding, reliable messaging, secure session information, and proxy configuration. For details, see "Advanced Scenario Setting" on page 1092.
Scenario type	The security scenario type: No scenario or a sub-type of Core, Security, WCF, or Optimization scenarios.

Select Certificate Dialog Box

This dialog box enables you to search and locate a certificate from a file or Windows store.

To access	<p>Select a scenario that uses a certificate in one of the following ways:</p> <ul style="list-style-type: none"> ▶ Open the Security Scenario Editor: Choose SOA Tools > Security Scenario Editor. ▶ In the Manage Services dialog box, select the Protocol and Security tab and click the Edit Data button. <p>Select a WCF scenario that uses a client or service certificate, such as WsHttpBinding or Federation. In the Certificate field, click the Browse button.</p>
Important information	<p>This only applies to security scenarios that allow you to specify client, server, or service certificates.</p>
Relevant tasks	<ul style="list-style-type: none"> ▶ "Create a scenario (if you do not have existing ones)" on page 1109. ▶ "Load a security scenario (if you have existing ones)" on page 1110.

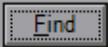
Select Certificate from File

When you choose **File**, the dialog box shows the user interface elements are described below:

UI Elements	Description
	Browse. Allows you to locate the certificate file with a .pem, .arm, .der, or .pfx extension.
File	The complete path of the certificate file.
Password (optional)	The password required to access the certificate.

Select Certificate from Windows Store

When you choose **Windows Store**, the dialog box shows the user interface elements are described below:

UI Elements	Description
	Begins the search for the certificate.
Import from	The location of the certificate: <ul style="list-style-type: none"> ➤ Windows store ➤ File
Store location	The store location, for example Current User .
Store name	The store name, for example, AuthRoot .
Search text	The text to match in the certificate name.
Password (optional)	The password required to access the certificate.
<certificate list>	A list of the certificates in the Windows store sorted by Subject, Issuer, Private, Store Location, and Store Name.

Web Services Security Examples

This section illustrates several common security scenarios.

Authenticating with a Username Token

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

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

Signing a Specific Element with an X.509 Certificate

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

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

Signing with an X.509 Certificate

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

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

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

Encrypting with a Certificate

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

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

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

Since we want to generate a Subject Key Identifier, we set the Add value to **False**. For more information, see "Reference a token with a SubjectKeyIdentifier - optional" on page 1105.

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);
```

Tips and Guidelines

This section provides a "how to" guide for testing WCF services and defining security scenarios.

WCF Guidelines

This section describes how to use VuGen for 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 Categories" on page 1084. 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 Categories" on page 1084. 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 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**).

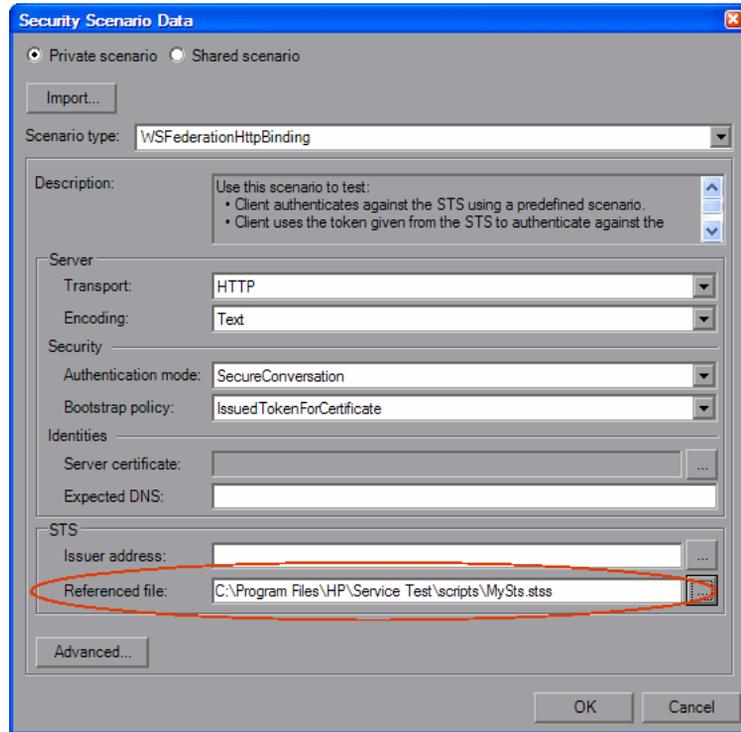
Choose the scenario **WCF > WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

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.

- 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 Testing

This section provides a summary of using Service Test for general security testing.

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 Scenario Setting" on page 1092. In case a very low-level control over the WS-Security tokens is required use the legacy security model. For more information, see the "Setting Security Overview" on page 1076.

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**).

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.

Tips for Advanced Standards Testing

This section provides a quick summary of using VuGen for advanced standards testing.

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 Scenario Setting" on page 1092.

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 Scenario Setting" on page 1092.

39

Web Services - Service Emulation

This chapter includes:

Concepts

- ▶ Emulated Services Overview on page 1130

Tasks

- ▶ How to Create an Emulated Service – Workflow on page 1139

Reference

- ▶ Service Emulation Console User Interface on page 1142

Troubleshooting and Limitations on page 1158

Concepts

Emulated Services Overview

The **Service Emulation Console** helps you create an emulation of a service in order to test other Web services in your environment.

Note: The Service Emulation tool is only available with a Service Test license. For details, contact HP Support.

The emulated service provides the following benefits:

- ▶ **Early stage development.** It lets you design and run tests at early stages of development when the actual service is inaccessible. For example if the development of the service is incomplete or if the service's host is unavailable, you can use an emulated service to test other services in your application.
- ▶ **Client testing.** Using the Service Emulator, you can test the functionality of your client application.
- ▶ **Isolating components.** If the service you want to test is part of a complex system, you can use the Service Emulator to test one service without its dependencies. The dependent services may be incomplete, temporarily unavailable, or simply distracting to your testing plans.

The Service Emulation console enables you to define the service's behavior through delays and rules.

You create a service by specifying a WSDL file that defines the operations and parameters of the service. When you specify a WSDL file, the Service Emulation tool uses the current structure of the WSDL to define the structure of the service's input and output data.

If the original WSDL changes, it will not be reflected in your emulated service. To use an updated WSDL, recreate the emulated service.

This section includes:

- "Server Emulation Hosts" on page 1131
- "Emulation Service Behavior" on page 1131
- "Client Testing" on page 1136

Server Emulation Hosts

The **host** is the machine to which you send emulated service requests. The emulation server is an Apache Tomcat server, installed during HP Service Test setup. You can also specify other machines, upon which a Tomcat server is installed.

For information about selecting a host, see "Add a host" on page 1139.

For information about starting and stopping the emulation server, see "Activate the server" on page 1139.

Emulation Service Behavior

The Service Emulation console lets you specify a behavior for each operation. You specify the behavior of the service's operations by:

- Delaying an Operation's Response
- Providing a Default Response
- Setting Service Emulation Rules

Delaying an Operation's Response

You can set a time delay after which the server will respond.

To set a global delay for the entire operation, see "Providing a Default Response" on page 1132.

To set a delay for a specific rule, see "Response Pane" on page 1147.

Providing a Default Response

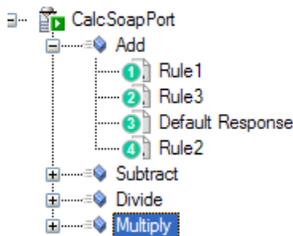
The default response is the set of values that the operation will use, if no rules are present.

You can manually specify the values for the default response, or you can import them from an XML file containing sample results.

Setting Service Emulation Rules

In addition to setting the default response, you can also set behavior rules. Through rules, you define a distinct behavior for the service—the expected response based on the request, input data.

You can set multiple rules for your operation. You arrange the rules in order of priority. If there is a conflict between rules, the emulated service follows the position of the rules. A number icon indicates the priority of the rule. In the following example, **Rule 1** has the highest priority. For information on changing the rule priority, see the "Console" on page 1142.



You can set two type of behavior rules: XML rules or Code rules.

XML Rules

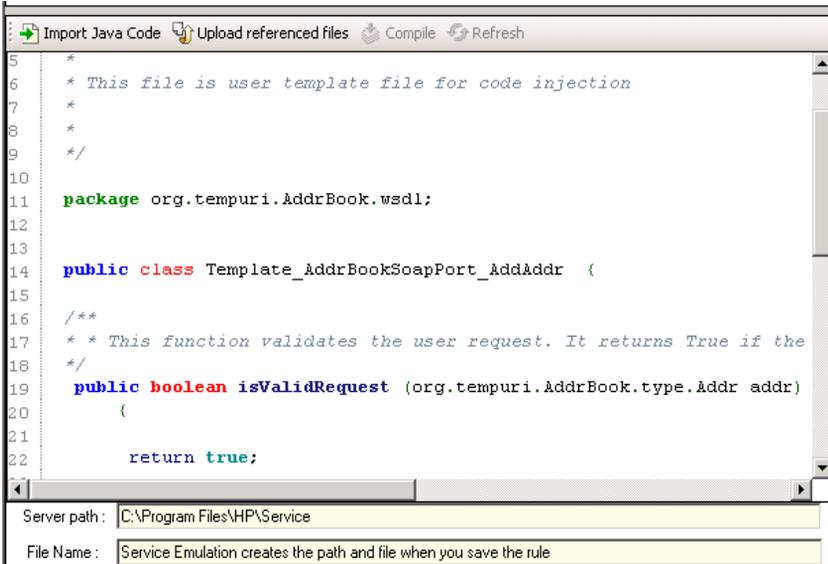
XML rules are rules in which you assign constant values to the request and/or response XML elements. You can set the values manually, or by importing an existing XML SOAP file. For example, for an Add operation, you could specify 4 as the first argument, 5 as the second argument, and a result of 9.

You can exclude arguments from the rule definition. This is useful if you want your service to return a particular response, ignoring the value of one of the arguments. For example, for a multiplication operation, you can set a rule indicating that if the first argument is 0, the result will be 0, regardless of the value of the second argument. For more information, see "Define XML rules - optional" on page 1140.

Code Rules

Code rules are small snippets of code that allow you to program the response values of the emulated service. You can use conditional statements to determine whether the particular rule is applicable, and respond accordingly.

Code rules let you use standard Java code and classes to determine whether the particular rule should be applied.



```

5  *
6  * This file is user template file for code injection
7  *
8  *
9  */
10
11 package org.tempuri.AddrBook.wsdl;
12
13
14 public class Template_AddrBookSoapPort_AddAddr {
15
16     /**
17     * * This function validates the user request. It returns True if the
18     */
19     public boolean isValidRequest (org.tempuri.AddrBook.type.Addr addr)
20     {
21
22         return true;

```

Server path : C:\Program Files\HP\Service
File Name : Service Emulation creates the path and file when you save the rule

The code uses two primary methods: **isValidRequest** and **getResponse**. The **isValidRequest** method checks the values of the input arguments and **getResponse** calculates and returns the response.

If **isValidRequest** returns **true**, the Service Emulation uses this rule and returns a response using **getResponse**. If **isValidRequest** returns **false**, it moves on to the next rule in the tree or, if there are no other rules, the Service Emulation uses the Default response.

Code Rule Examples

In the following example, the Code rule checks the values of the request. If the values are **1** and **2**, it instructs the service to use the sum of the values as the response.

```
/**
 * Service: CalcSoapPort
 * Operation: Add
 *
 * This file is user template file for code injection
 */
package org.tempuri.Calc.wsdl;

public class Template_CalcSoapPort_Add {

    public boolean isValidRequest (double a, double b) throws java.rmi.RemoteException
    {

        if ((a == 1) && (b ==2))
            return true;
        else
            return false;

    }

    public double getResponse (double a, double b) throws java.rmi.RemoteException
    {
        return (a+b);
    }
}
```

The next example shows a similar snippet that uses external classes for the response. When using references classes, you need to manually import them to the server using the **Upload Referenced Files** button.

```
package org.tempuri.Calc.wsdl;

public class Template_CalcSoapPort_Add {

    public boolean isValidRequest (double a, double b) throws java.rmi.RemoteException
    {

        if ((a == 1) && (b ==2))
            return true;
        else
            return false;

    }

    public double getResponse (double a, double b) throws java.rmi.RemoteException
    {

        double resp = -1;

        try {
            MyTestAdd adr = new MyTestAdd();
            adr.SetNumbers (10,20);

            resp=adr.GetAdrResponse();

            return resp;
        }
        catch(Exception ex) {
            System.out.println ("getResponse Exception:" + ex);
        }
        return resp;
    }
}
```

For information on setting up code rules, see "Define code rules - optional" on page 1141.

Client Testing

The **host** is the machine to which you send emulated service requests. The emulation server is an Apache Tomcat server, installed during HP Service Test setup. You can also specify other machines, upon which a Tomcat server is installed.

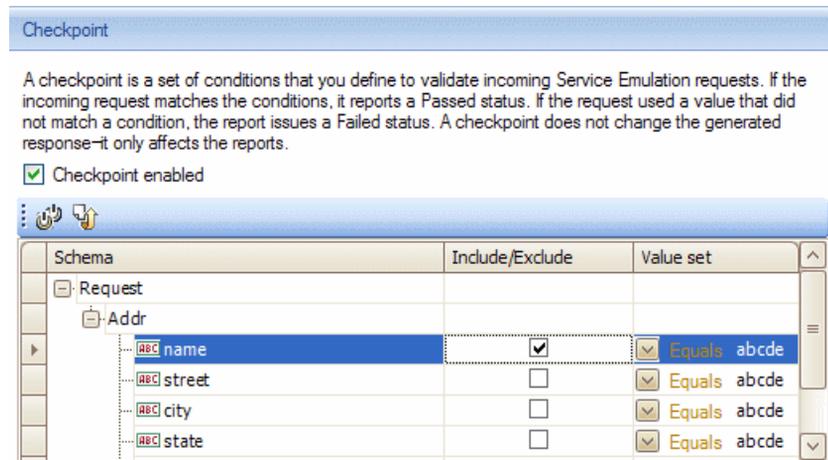
For information about client testing, see "Configure the client testing tools - optional" on page 1141.

You can perform client testing in the following areas:

- Checkpoints
- Labels
- Service Emulation Reports

Checkpoints

A Service Emulation checkpoint is a set of conditions that you define, to validate incoming requests. The conditions are the argument values together with comparison operators, with which you expect the request to comply. For example, you can set a checkpoint to verify that the value of argument **A** is greater than or equal to 2.



If the emulated service's incoming request matches the conditions, it reports a **Passed** status. If, however, the request used a value that did not match a condition, the report issues a **Failed** status.

A checkpoint does not change the generated response.

You can set a checkpoint for an entire operation or for a specific rule. If you define a checkpoint for both an operation and a rule below it, the incoming request will need to comply with both checkpoints to be considered **Passed**.

A rule checkpoint is only activated if its parent rule is activated. This means that a rule's checkpoint will only be validated when the request complies with the rule.

For information about checkpoints, see "Enable and define checkpoints - optional" on page 1141.

Labels

Labels allow you to put a date and time stamp on service calls sent to the emulation server.

These labels will allow you to filter you data when generating reports. For more information, see the "Configure the client testing tools - optional" on page 1141.

Service Emulation Reports

You can create reports to view, filter and sort the information about your emulated service. You select which information to include and the display order. You can filter by date or label, service, operation, checkpoint status, client IP, and Session ID.

For more information, see the "Configure the client testing tools - optional" on page 1141.

Clearing the Log

Service Test allows you to clear the database log, containing the service calls sent through the emulated services. You can indicate the range of entries to remove by either date or label. The custom defined settings such as rules, checkpoints, and default responses, remain unaffected.

For more information, see the "Configure the client testing tools - optional" on page 1141.

Tasks

How to Create an Emulated Service – Workflow

This task describes the workflow for creating an emulated service.

This task includes the following steps:

- "Add a host" on page 1139
- "Activate the server" on page 1139
- "Create a new emulated service" on page 1140
- "Set the default behavior - optional" on page 1140
- "Define XML rules - optional" on page 1140
- "Define code rules - optional" on page 1141
- "Enable and define checkpoints - optional" on page 1141
- "Integrate the emulated service" on page 1141

1 Add a host

Specify a host to which you will be sending the Web Service calls. For more information, see "Host Selection Dialog Box" on page 1150.

2 Activate the server

Make sure the Service Test Emulation server is active on the designated host machine. By default, Service Test automatically invokes the Tomcat emulation server when it opens.

To check if the server is active:

Enter the following URL into your browser:

`http://<hostname>:8080/ServiceEmulation/index.jsp`.

If the server is active, the browser will display **HP Service Emulation**.

To start the server:

Choose **Start > All Programs > LoadRunner > Service Test > Start Emulation Service**.

To stop the server:

Choose **Start > All Programs > LoadRunner > Service Test > Stop Emulation Service**.

For troubleshooting information about the server, see "Troubleshooting and Limitations" on page 1158.

3 Create a new emulated service

Specify a file or URL for a WSDL and select a host for this service. For more information, see the "New Emulated Service Dialog Box" on page 1152.

4 Set the default behavior - optional

Set the default response for the service, to use when there are no relevant rules. For more information, see the "Details Pane - Default Response" on page 1145.

5 Define XML rules - optional

Define one or more XML rules for each operation to set the rule's behavior. To add an XML rule, select an operation and choose **New XML Rule** from the shortcut menu.

- Set the rule for the request as described in "Request Pane" on page 1147.
- Set the details for the response as described in "Response Pane" on page 1147.

6 Define code rules - optional

Define one or more code rules for each operation using the built-in Java template and modifying the code for your needs. To add a code rule, select an operation and choose **New Code Rule** from the shortcut menu. Save the rule after you make changes. For more information, see "Details Pane - Code Rule" on page 1148.

7 Enable and define checkpoints - optional

Define one or more checkpoints per operation or per rule. To add a checkpoint, select the entity (operation or rule) and choose **New Checkpoint** from the shortcut menu. For more information, see the "Details Pane - Checkpoint" on page 1149.

8 Configure the client testing tools - optional

Perform client testing using the following tools:

- ▶ **Define labels.** See the "New Label Dialog Box" on page 1157.
- ▶ **Generate reports.** See the "Service Emulation Report Wizard" on page 1153.
- ▶ **Clear the Web Service log.** See the "Clear Service Call Log Dialog Box" on page 1156.

9 Integrate the emulated service

After you create an emulated service, you incorporate it into your script for testing purposes.

- a** In the Service Emulation console, select a service in the left pane. Copy the **WSDL location** from the **Emulated service** section to the clipboard.
- b** Open the **Manage Services** dialog box and select a service. Select the **Override Address** option. Paste the WSDL location into the **Service Address** box. During the test run, VuGen will send service requests to that location.

Reference

Service Emulation Console User Interface

This section includes the elements of the console (in alphabetical order):

- ▶ Console on page 1142
- ▶ Host Selection Dialog Box on page 1150
- ▶ Host Configuration Dialog Box on page 1151
- ▶ Host Name Pane on page 1151
- ▶ New Emulated Service Dialog Box on page 1152
- ▶ Service Emulation Report Wizard on page 1153
- ▶ Clear Service Call Log Dialog Box on page 1156
- ▶ New Label Dialog Box on page 1157

Console

The Service Emulation Console lets you create emulated services and define their behavior.

To access	Start > All Programs > LoadRunner > Service Test > Service Emulation Console
-----------	---

Important information	The Service Emulation tool uses an SQL Server database to store the rules and their expected values. For LoadRunner with Service Test, you need to manually install the redistributable version of SQL Microsoft SQL Server Management Studio Express 9.00. Run the installation file, install_MSSQL2005Express.bat , located in the Additional Components\MSSQL2005Express folder on the HP LoadRunner media.
Relevant tasks	<ul style="list-style-type: none"> ➤ "Set the default behavior - optional" on page 1140 ➤ "Define XML rules - optional" on page 1140 ➤ "Define code rules - optional" on page 1141 ➤ "Enable and define checkpoints - optional" on page 1141

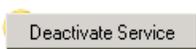
User interface elements are described below:

UI Elements	Description
	Add New Host. Opens the Host Selection dialog box
	New Emulated Service. Opens the New Emulated Service dialog box.
	New XML Rule. Opens a grid in the right pane to let you define a new XML rule
	New Code Rule. Opens an editor in the right pane to let you enter a new code-based rule
	New Checkpoint. Opens the Checkpoint grid in the right pane
	<p>Save. Updates the database with all rules and modifications.</p> <p>Tip: An asterisk in the title bar indicates that your service has not been saved.</p>
	Refresh. Refreshes the WSDL from its original location
	Delete. Removes the selected entity: emulated service, rule or checkpoint

UI Elements	Description
	Activate. Activates a service
	Deactivate Service. Temporarily deactivates a service
	Move Rule Up. Moves a rules up to raise its priority
	Move Rule Down. Moves rules down to lower their priority
	Add label. Adds a label to the service for filtering purposes.
	Generate Report. Opens the Generate Report wizard
	Clear Service Call Log. Lets you clear service calls from the database log sent during a specific range of dates.
<Host Name pane>	A list of all the defined hosts. Default value: localhost
<Details pane>	Details of the entity selected in the left pane: Service, Operation, Default Response, XML Rule, Code Rule, or Checkpoint

Details Pane - Services

User interface elements are described below:

UI Elements	Description
	Activates (only) the service selected in the left pane
	Deactivates the service selected in the left pane (only).
Name/ Package	The name and package of the emulated service.
Original Service/ WSDL location	The path or URL from where the original WSDL was imported.

UI Elements	Description
Emulated Service/ WSDL location	The URL of the service on the machine running the emulation server.
Emulated Service/ Endpoint location	The endpoint to which you want to send the request

Details Pane - Operation

User interface elements are described below:

UI Elements (A-Z)	Description
Name	Operation name (read-only)
Comments	An editable field with comments about the operation.

Details Pane - Default Response

The service emulator returns the default response in the case that no rules are present or that none of the rules match the request. User interface elements are described below:

UI Elements	Description
	Add column. Adds a new Value Set column.
	Remove. Removes the selected Value Set column.
	Reset. Discards your changes in the selected value set, and uses the WSDL's default response values.
	Import SOAP. Opens the Select File counting SOAP message dialog box to load values from a SOAP message.
Response type	The type of SOAP response: Response or Fault .

UI Elements	Description
Response delay (sec.)	The type of delay to introduce before the server response: Constant or Random . For a constant delay, specify the delay in seconds. For a random delay, specify a range of time. Tip: To indicate milliseconds, use the values to the right of the decimal. For example, for 20 msec., enter 0.02.
Scrolling mode	Freeze schema column. Freezes the schema column, allowing you to scroll freely through the value sets.
Schema pane	A schema of the service's response: <ul style="list-style-type: none"> ▶ Schema: an XML representation of the elements ▶ Value set: A set of values for each of the arguments. Use the Add Column button to add more value sets. Enter the values manually or use the Import Soap button to load values from a SOAP message.
Value Selection	Value selection method: Sequential or Random .

Details Pane - XML Rule

The service emulator returns the default response in the case that no rules are present or that none of the rules match the request. User interface elements are described below:

UI Elements	Description
Name	A name for the XML rule as it will appear in the left pane's tree hierarchy and in the report.
Request Pane	Details about the request to send to the emulation server. See " Request Pane " on page 1147.
Response Pane	Details about the response to receive from the emulation server. See " Response Pane " on page 1147.

Request Pane

UI Elements	Description
	Reset. Discards your changes in the selected value set, and uses the WSDL's default response values.
	Import SOAP. Opens the Select File containing SOAP message dialog box to load values from a SOAP message.
	Data type. Shows an argument's data type during mouseover.
	Comparison operator list. <ul style="list-style-type: none"> ▶ For numbers and dates: Equals, GreaterOrEqual, LessOrEqual, GreaterThan, or LessThan. ▶ For strings: Equals, Not Equal, Contains, EndsWith, or StartsWith. Tip: You can also use regular expressions
Schema column	A schema of the XML elements
Include/Exclude column	Indicates whether to include or exclude the argument in the request to the server.
Value set column	A set of values and comparison operators for each of the arguments. Choose comparison operators and specify a value after determining the data type of the argument.

Response Pane

UI Elements	Description
	Add column. Adds a new Value Set column.
	Remove. Removes the selected Value Set column.
	Reset. Discards your changes in the selected value set, and uses the WSDL's default response values.
	Import SOAP. Opens the Select File containing SOAP message dialog box to load values from a SOAP message.
Response type	The type of SOAP response: Response or Fault .

UI Elements	Description
Response delay (sec.)	The type of delay to introduce before the server response: Default (value of Default response), Constant or Random . For a constant delay, specify the delay in seconds. For a random delay, specify a range of time. Tip: To indicate milliseconds, use the values to the right of the decimal. For example, for 20 msec., enter 0.02.
Scrolling mode	Freeze schema column. Freezes the schema column, allowing you to scroll freely through the value sets.
Schema pane	A schema of the service's response: <ul style="list-style-type: none"> ▶ Schema: an XML representation of the elements ▶ Value set: A set of values for each of the arguments. Use the Add Column button to add more value sets. Enter the values manually or use the Import Soap button to load values from a SOAP message.
Value selection	Value selection method: Sequential or Random .

Details Pane - Code Rule

The Service Emulation Code Rule area lets you edit java-baser rule to apply to your service. User interface elements are described below:

UI Elements	Description
 Import Java Code	Import Java Code. Copies an existing Java file into the editor pane. Edit the file as required. Note: Imported Java code must be in the same package as the code of the code rule.
 Upload referenced files	Upload Referenced File. Enables you to upload external classes referenced by your code.
 Compile	Compile. Compiles the code on the server to check its validity.
 Refresh	Refresh. Synchronizes the code if it was modified on the server machine.

UI Elements	Description
<user template>	A template containing the code to apply to the emulated service. Note: Your code can adversely affect the application. Do not use code that will interfere with or stop the Service Emulation service, such as System.exit(0) . Do not remove or modify the names or parameters of the isValidRequest and getResponse built-in methods.
File Name	An automatically generated file with a .java extension for the code rule.
Server Path	The location of the server.

For information about complex input arguments, see the server notes at <http://ws.apache.org/axis/java/apiDocs/javax/xml/rpc/holders/Holder.html>.

Details Pane - Checkpoint

The Service Emulation checkpoint area lets you set conditions to validate incoming requests. You can apply a checkpoint to an entire operation or to an individual rule. User interface elements are described below:

UI Elements	Description
	Reset. Discards all changes in the value set column.
	Import SOAP. Opens the Select File containing SOAP message dialog box to load values from a SOAP message.
	Data type. Shows an argument's data type during mouseover.
	Comparison operator list. <ul style="list-style-type: none"> ▶ For numbers and dates: Equals, GreaterOrEqual, LessOrEqual, GreaterThan, or LessThan. ▶ For strings: Equals, Not Equal, Contains, EndsWith, or StartsWith. Tip: You can also use regular expressions

UI Elements	Description
Checkpoint enabled	Enables the checkpoint verification mechanism.
Schema column	A schema of the XML elements.
Include/Exclude column	Indicates whether to include or exclude the argument in the request to the server.
Value set column	A set of values and comparison operators for each of the arguments. Choose comparison operators and specify a value after determining the data type of the argument.

Host Selection Dialog Box

Enables you to select a host machine and port for your emulated service.

To access	File > New Host or the New Host button
Important information	The host machine must have the Tomcat server. This is installed automatically on the localhost.
Relevant tasks	"Add a host" on page 1139

The following elements are included:

GUI Element (A-Z)	Description
	Browse. Opens the Browse for Computer dialog box
Host Name	The name or IP address of the machine to host the emulated service
Port	The port number for the specified host machine Default value: 8080 Tip: If you have a conflict with the 8080 port, edit the Server.xml file in the apache-tomcat-5.5.17\conf folder under the Service Test installation. Change the port from 8080 to another port. Create a new localhost server with the new port number.

Host Configuration Dialog Box

Enables you to view the host name and set the database connection settings.

To access	In the Host name toolbar, in the left pane, click the Host Configuration button  .
Important information	If your database requires a username and password, you need to enter them here.
Relevant tasks	"Add a host" on page 1139
See also	"Host Selection Dialog Box" on page 1150

The following elements are included:

GUI Element (A-Z)	Description
Server	hostname\instance of the database
Port	port on the database server used by the emulated service
Username\Password	credentials for logging into the database server Default value: empty for localhost

Host Name Pane

Displays the host name and lets you start or stop the server, and open the Host Configuration dialog box

To access	Automatically displayed in the left pane.
Important information	The lower part of the left pane contains a toolbar for each of the available hosts.
Relevant tasks	"Activate the server" on page 1139
See also	<ul style="list-style-type: none"> ➤ "Host Selection Dialog Box" on page 1150 ➤ "Host Configuration Dialog Box" on page 1151 ➤ "Console" on page 1142

The following elements are included (unlabeled GUI elements are shown in angle brackets>):

GUI Element	Description
	Start Emulation Server. Starts the Tomcat server and invokes the ServiceEmulationClient process.
	Stop Emulation Server. Stops the Tomcat server and the ServiceEmulationClient process.
	Configure Host. Opens the Host Configuration dialog box to set the database connection settings.
<hostname>	the machine hosting the displayed emulated services Default value: localhost

New Emulated Service Dialog Box

This dialog box lets you define an emulated service by indicating a WSDL and selecting a host.

To access	Service > New Emulated Service
Important information	Importing is not supported for WSDLs that import schemas. To import this type of WSDL, deploy it on a local server and then import it using the generated URL.
Relevant tasks	"Create a new emulated service" on page 1140
See also	"Host Selection Dialog Box" on page 1150

User interface elements are described below:

UI Elements (A-Z)	Description
	Browse. Opens a browser (for URL selection) or the Open file dialog box (for File selection)
	Opens the Host Selection Dialog Box.

UI Elements (A-Z)	Description
Select host	A dropdown list of the available host machines.
Select WSDL from	The source of the WSDL: <ul style="list-style-type: none"> ➤ URL ➤ File

Service Emulation Report Wizard

This wizard enables you to generate reports about your emulated service.

To access	Client Testing > Generate Report
Relevant tasks	"Configure the client testing tools - optional" on page 1141
Wizard map	This wizard contains: Welcome > Select Columns Page > Filtering Page > Sorting Page
See also	"New Label Dialog Box" on page 1157

Select Columns Page

This wizard page enables you to select the column to include in the report.

Important information	General information about this wizard is available here: "Service Emulation Report Wizard" on page 1153.
Wizard map	The Service Emulation Report Wizard contains: Welcome > Select Columns Page > Filtering Page > Sorting Page

User interface elements are described below:

UI Elements (A-Z)	Description
All fields	All of the available fields of the report: <ul style="list-style-type: none"> ▶ Client IP Address ▶ Date and time ▶ Operation ▶ Rule ▶ Service ▶ Session ID
Fields to be displayed	The fields to appear in the report in the order in which they appear. <ul style="list-style-type: none"> ▶ To change the order of the columns in the report, use the Up  and Down  buttons in the Fields to be displayed pane. ▶ To add or remove fields from the report, use the horizontal arrow buttons: <ul style="list-style-type: none"> ▶  . Add selected field to report. ▶  . Add all fields to report. ▶  . Remove selected field from report. ▶  . Remove all fields from report.

Filtering Page

This wizard page enables you to select the column to include in the report.

Important information	General information about this wizard is available here: "Service Emulation Report Wizard" on page 1153.
Wizard map	The Service Emulation Report Wizard contains: Welcome > Select Columns Page > Filtering Page > Sorting Page

User interface elements are described below:

UI Elements	Description
From	Lower limit for the report generation through a time or label. For more information, see Labels. Tip: Disable option to include all data from the beginning of the service call log.
To	Upper limit for the report generation through a time or label. For more information, see Labels. Tip: Disable option to include all data until the end of the service call log.
Service	Services to include in the report. Select each service individually. To show all, disable the option.
Operation	Operations to include in the report. Select each operation individually. To show all, disable the option.
Checkpoint Status	The statuses to show: Failure , Success , and Undefined . To show all, disable the option.
Client IP address	The client IP addresses to include in the report. To show all, disable this option.
SessionID	The session IDs to include in the report. To show all, disable this option.

Sorting Page

This wizard page enables you to select the column to include in the report.

Important information	General information about this wizard is available here: "Service Emulation Report Wizard" on page 1153.
Wizard map	The Service Emulation Report Wizard contains: Welcome > Select Columns Page > Filtering Page > Sorting Page

User interface elements are described below:

UI Elements (A-Z)	Description
Asc/Desc	The order in which to display the data in the report. The sorting order is not global and may be set differently for each item. Default: Ascending order, for each item.
All fields	All fields by which the report can be sorted: Date and time, Service, Operation, Client IP address, and Session ID.
Ordering fields	The fields by which to sort the report data. Use right facing arrows to include an item in the sorting list. Use the up and down arrows to set priorities in the ordering.

Clear Service Call Log Dialog Box

This dialog box enables you to clear the service call log in the database. You can clear the whole log or a part of it, based on dates or labels.

To access	Client Testing > Clear Service Log >
Relevant tasks	"Configure the client testing tools - optional" on page 1141

User interface elements are described below:

UI Elements (A-Z)	Description
From	Lower limit for the report generation through a time or label. For more information, see Labels. Tip: Disable this option to clear all data from the beginning of the log until the <i>To</i> threshold.
To	Upper limit for the report generation through a time or label. For more information, see Labels. Tip: Disable this option to clear all data from the <i>From</i> threshold until the end of the log.

New Label Dialog Box

This dialog box enables you to put a date and time stamp on incoming emulated service calls. These labels will allow you to filter you data when generating reports.

To access	Client Testing > Add Label
Relevant tasks	"Configure the client testing tools - optional" on page 1141
See also	<ul style="list-style-type: none"> ➤ "Service Emulation Report Wizard" on page 1153 ➤ "Clear Service Call Log Dialog Box" on page 1156

User interface elements are described below:

UI Elements (A-Z)	Description
Current time	Time stamp associated with the label (read-only).
Label name	The label name as it will appear in the reports and filtering options.

Troubleshooting and Limitations

This section provides troubleshooting information for the emulated service and its database.

Changing Port Numbers

If the standard port is not available, you can modify emulated service port by changing the port number in the configuration file. The configuration file, **httpd.conf**, is stored in Service Test's **apache/conf** directory.

To modify the port number:

- 1** Open the **httpd.conf** file with any text editor.
- 2** Modify the entry "Listen 80" to the desired port number, for example "Listen 8080. "
- 3** Modify the entry which contains "ServerName localhost:80" to indicate the desired port, for example "ServerName localhost:8080."
- 4** Restart the Tomcat server. Select **Programs > Start > HP Service Test > Start Emulation Server**.

Cannot Open Service Emulation Service on Computer

If the Service Emulation console indicates that the server is not accessible, even after a manual start, you can try the following troubleshooting tips:

- Make sure the server is up. Enter the following URL into your browser: <http://localhost:8080/ServiceEmulation/index.jsp>. If the server is down, start it manually from the Start menu.
- Verify that port 8080 is available. If it is not, release the port and restart the server. Alternatively, edit the **Server.xml** file in the **apache-tomcat-5.5.17\conf** folder under the Service Test installation. Change the port from 8080 to another port. Create a new localhost server with the new port number.
- Open the Apache Tomcat log files under the `<product install dir>\Service Test\apache-tomcat-5.5.17\logs` directory and determine the reason that the server did not load. Fix the problem and reload the server.

- ▶ When emulating a service, if you encounter a warning "Service is not activated" in the endpoint location of your emulated service, perform one or more of the following actions:
 - ▶ Verify that the service is active. Select the service in the left pane and click the Activate Service button in the right pane, if it is visible.
 - ▶ If the service is already active, check its URL. Copy the URL from the WSDL Location in the Emulated Service's right pane, and paste it into a browser, removing the suffix `?wsdl` from the string. For example, instead of `http://localhost:8080/axis/services/MyService?wsdl`, use `http://localhost:8080/axis/services/MyService`. If your browser opens a valid page, then your service is active. To use this emulated service, import the original WSDL into Service Test. Then override the address and set it to the URL you previously used in the browser.

For more information about integrating your emulated service into your test, see "Integrate the emulated service" on page 1141.

Database Server Connection Issues

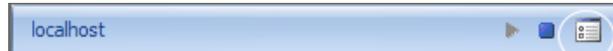
The Service Emulation stores its data in an MS SQL database, installed with Service Test.

If you are unable to connect to the database, (For example, if you attempt to activate an emulated service, and the console indicates that it is unable to connect to the remote server), follow these steps:

To troubleshoot your server connection:

- 1** Verify that SQL Server 2005 is installed on the server host. This installation is a prerequisite and is usually included during setup.
- 2** From the **Start** Menu, choose **Microsoft SQL Server 2005 > Configuration Tools > SQL Server Configuration Manager**. Click the **SQL Server 2005 Services** node, and verify that there is an instance of SQL Server, for example, **SQLEXPRESS**.
- 3** Expand the **SQL Server 2005 Network Configuration** node, and make sure that the **TCP/IP** is set to **Enabled** for the running instance. If it is not enabled, use the right-click menu to enable it.

- 4 Double-click on **TCP/IP** to view the properties. Click the **IP Addresses** tab and note the port to which the instance is configured. If no port is specified, set a port number and then re-start the SQL Server instance.
- 5 Open the Service Emulation Console, and click the **Host Configuration** button on the right corner of the host (for example **localhost**) toolbar. For more information, see the "Host Configuration Dialog Box" on page 1151.



- 6 Edit the database connection settings to match the running SQL Server instance and the TCP/IP port number:

The setup uses **SQLEXPRESS** as the default instance, and **1433** as the default port. If the information from the Configuration Manager is different, edit the values in the dialog box accordingly. In the **Server** box, enter the **<host>\<instance>** combination. If an un-named instance is running, enter **localhost** in the **Server** box.

40

Windows Sockets Protocol

This chapter includes:

Concepts

- ▶ Recording Windows Sockets Overview on page 1162

Tasks

- ▶ How to Record a WinSocket Script on page 1174
- ▶ How to View and Modify WinSocket Buffers on page 1176

Reference

- ▶ Data Buffers on page 1182
- ▶ Windows Socket User Interface on page 1186

Concepts

Recording Windows Sockets Overview

The Windows Sockets protocol supports applications which communicate over the TCP/IP protocol using a Microsoft WinSock DLL. The WinSock protocol allows you to see the actual data sent and received by the buffers.

The WinSock protocol records function that relate to the sockets, data buffers, and the Windows Sockets environment. Using VuGen, you record your application's API calls to the Winsock.dll or Wsock32.dll.

For example, you could create a script by recording the actions of a *telnet* application.

After creating a script, you can view the recorded buffers as raw data or as a snapshot. For details, see "Windows Socket Data" on page 1164 or "WinSock Data Buffer Snapshot" on page 1166.

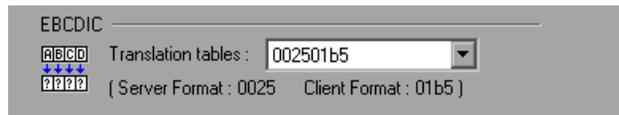
This section also includes:

- "Translation Tables" on page 1163
- "Windows Socket Data" on page 1164
- "WinSock Data Buffer Snapshot" on page 1166
- "Data Viewer" on page 1168
- "Data Navigation Tools" on page 1170
- "Buffer Data Editing" on page 1172

Translation Tables

You can display Windows Socket data in EBCDIC format through a translation table.

A translation table allows you to specify the format for recording when using the WinSock single protocol, and for code generation when using a WinSock multi protocol. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. If your data is in ASCII format, it does not require translation.



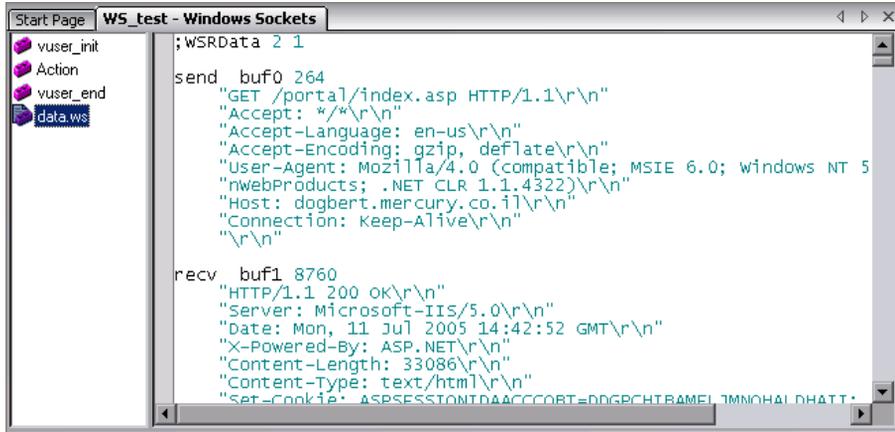
The first four digits of the listbox item represent the server format. The last four digits represent the client format. In the above example, the selected translation table is 002501b5. The server format is 0025 and the client format is 01b5 indicating a transfer from the server to the client. In a transmission from the client to the server, you would select the item that reverses the formats—01b50025 indicating that the client's 01b5 format needs to be translated to the server's 0025 format.

The translation tables are located in the **ebcdic** directory under the VuGen's installation directory. If your system uses different translation tables, copy them to the **ebcdic** directory.

For details on selecting a translation table in the recording options, see the "WinSocket Node" on page 413

Windows Socket Data

When you use VuGen to create a Windows Sockets Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **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's Script view to see the contents of the data file within the script editor.



```

;WSRData 2 1
send buf0 264
"GET /portal/index.asp HTTP/1.1\r\n"
"Accept: */*\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5
"nwebProducts; .NET CLR 1.1.4322)\r\n"
"Host: dogbert.mercury.co.il\r\n"
"Connection: Keep-Alive\r\n"
"\r\n"

recv buf1 8760
"HTTP/1.1 200 OK\r\n"
"Server: Microsoft-IIS/5.0\r\n"
"Date: Mon, 11 Jul 2005 14:42:52 GMT\r\n"
"X-Powered-By: ASP.NET\r\n"
"Content-Length: 33086\r\n"
"Content-Type: text/html\r\n"
"Set-Cookie: ASPSESSIONID=AACCCOBT=DDG6CHTBAMEI1JMMOHALDHATT:

```

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 bufindex number of bytes received
send bufindex

```

The buffer index begins with 0 (zero), and all subsequent buffers are numbered sequentially (1,2,3...) regardless of whether they are send or receive buffers.

In the following example, an `lrs_receive` function was recorded during a Vuser session:

```
lrs_receive("socket1", "buf4", LrsLastArg)
```

In this example, `lrs_receive` handled data that was received on `socket1`. The data was stored in the fifth receive record (`buf4`)—note that the index number is zero-based. The corresponding section of the `data.ws` file shows the buffer and its contents.

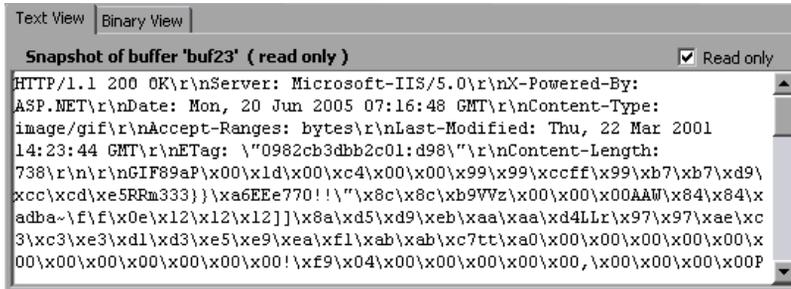
```
recv buf4 39
"\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
"\r\n"
"\r\n"
"SunOS UNIX (sunny)\r\n"
"\r"
"\x0"
"\r\n"
"\r"
"\x0"
```

For task details, see "How to View and Modify WinSocket Buffers" on page 1176.

WinSock Data Buffer Snapshot

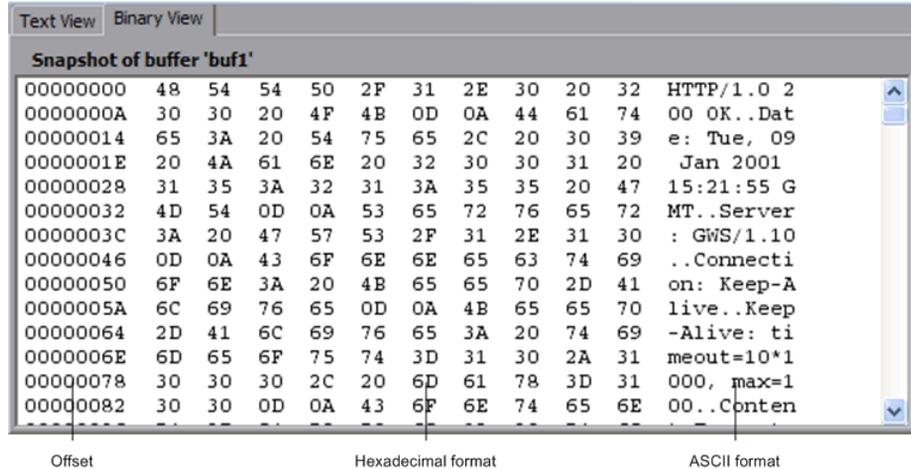
When viewing a Windows Socket script in tree view, VuGen provides a buffer snapshot window which displays the data in an editable window. You can view a snapshot in either **Text** view or **Binary** view.

The text view shows a snapshot of the buffer with the contents represented as text.



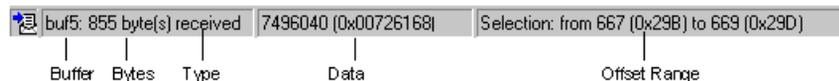
By default, VuGen stores the buffer data as read-only data. If you want to modify the contents of the buffer, clear the **Read only** box in the buffer's Text View. VuGen issues a warning that bookmarks and parameters may be affected.

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.



If you have data selected, the status bar also displays information about the data in Big Endian and Little Endian formats.

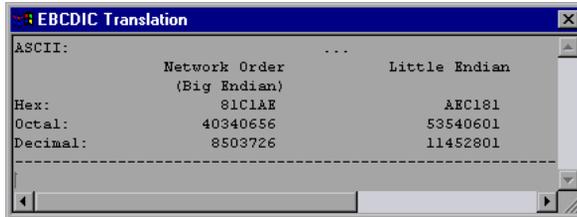


Data Viewer

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.



- If the selected text is more than four characters, VuGen displays the data in several columns in **long format**.

In the long 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.

Offset	Hexadecimal format	ASCII format
0	00 01 85 80 00 01 00 01 00 05 00 05 05 74 65 61tea
16	72 73 07 60 65 72 63 75 72 79 02 63 6F 02 69 6C	rs.mercury.co.il
32	00 00 01 00 01 C0 0C 00 01 00 01 00 00 03 84 00
48	04 C7 CB 4D 1A 07 6D 65 72 63 75 72 79 02 63 6F	..N..mercury.co
64	02 69 6C 00 00 02 00 01 00 00 03 84 00 09 06 6D	.il.....m
80	65 72 73 65 72 C0 35 C0 35 00 02 00 01 00 00 03	erser.5.5.....
96	84 00 02 C0 0C C0 35 00 02 00 01 00 00 03 84 005.....
112	08 08 7A 65 70 70 65 6C 69 6E C0 35 C0 35 00 02	..zeppelin.5.5..
128	00 01 00 00 03 84 00 0A 07 70 61 6C 65 72 6D 6Fpalermo
144	C0 35 C0 35 00 02 00 01 00 00 03 84 00 07 04 69	.5.5.....i
160	72 6F 6E C0 35 C0 4E 00 01 00 01 00 00 03 84 00	ron.5.N.....
176	04 C7 CB 4E 02 C0 0C 00 01 00 01 00 00 03 84 00	..N.....

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.

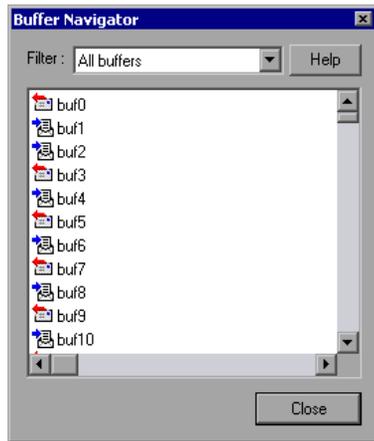
For details about customizing the short and long formats, see "Display Format" on page 1183.

Data Navigation Tools

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 Navigation

By default, VuGen displays all the steps and buffers in the left pane. The Buffer Navigator 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.

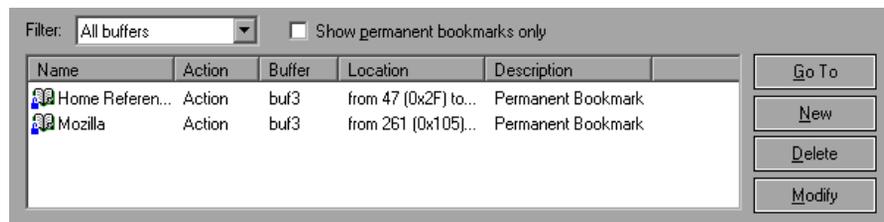
Note that you can also navigate between buffers by clicking on the buffer step in the left pane's tree view. The advantages of the buffer navigator are that it is a floating window with filtering capabilities.

Jump to Location

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.

Bookmarking

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.

Buffer Data Editing

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.

Text View		Binary View	
Buffer Snapshot			
00000000	48 54 54 50 2F 31 2E 31 20 32	HTTP/1.1 2	
0000000A	30 30 20 4F 4B 0D 0A 53 65 72	00 OK .Ser	
00000014	76 65 72 3A 20 4D 69 63 72 6F	ver: Micro	
0000001E	73 6F 66 74 2D 49 49 53 2F 34	soft-IIS/4	
00000028	2E 30 0D 0A 43 6F 6E 74 65 6E	.0..Conten	
00000032	74 2D 4C 6F 63 61 74 69 6F 6E	t-Location	
0000003C	3A 20 68 74 74 70 3A 2F 2F 64	: http://d	
00000046	6F 67 62 65 72 74 2F 49 6E 64	ogbert/Ind	
00000050	65 78 2E 68 74 6D 6C 0D 0A 44	ex.html..D	
0000005A	61 74 65 3A 20 57 65 64 2C 20	ate: Wed,	

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.

```
00000014 4F 4B 76 65 72 3A 20 4D 69 63 OKver: Mic
```

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:

```
00000014 31 39 32 37 39 76 65 72 3A 20 19279ver:
```

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:

```
00000014 30 78 34 42 34 46 76 65 72 3A 0x4B4Fver:
```

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.

For task details, see "Copy and paste blocks of data - optional" on page 1180.

Tasks

How to Record a WinSocket Script

This task describes how to set up a Windows Socket recording and how to record the session.

This task includes the following steps:

- "Open the recording options - optional" on page 1175
- "Select a translation table - optional" on page 1175
- "Exclude any non-relevant sockets - optional" on page 1175
- "Set a think time threshold - optional" on page 1176
- "Record the session" on page 1176
- "Parameterize the script - optional" on page 1176
- "Regenerate the script - optional" on page 1176

1 Open the recording options - optional

After creating a WinSock script, select **Tools > Recording Options** and click the **WinSocket** node.

2 Select a translation table - optional

In the **EBCDIC** section, select a translation table. If your data is in ASCII format, select the **None** option—otherwise VuGen will convert the ASCII data. For details, see "Translation Tables" on page 1163.

3 Exclude any non-relevant sockets - optional

In the **Exclude Settings** section, add any non-relevant sockets to the list. You should exclude hosts and ports that do not influence the server load under test, similar to the local host and the DNS port (53), which are excluded by default.

To exclude the entries from the recording, but include them in the log, clear the **Do not include excluded sockets in log** option.

For user interface details, see the "WinSocket Node" on page 413.

4 Set a think time threshold - optional

Indicate a think time threshold. If VuGen detects a pause in action less than the threshold time, it will not generate a **Think Time** step/**lr_think_time** function. For details, see the "WinSocket Node" on page 413.

5 Record the session

Record the session and save the script.

6 Parameterize the script - optional

Replace recorded values with parameters using the shortcut menu. For more information, see Chapter 9, "Parameters."

7 Regenerate the script - optional

If you need to regenerate the script, for example if you want to include an excluded host:port, or if the translation was not correct, choose **Tools > Regenerate**. In the WinSocket node, modify the settings.

Note: Script regeneration is only supported for multi-protocol scripts.

How to View and Modify WinSocket Buffers

The following steps describe how to view, modify, and navigate through WinSocket data.

- "View and modify the data in Script view - optional" on page 1177
- "Open the buffer in a viewer- optional" on page 1177
- "View and modify the data in Tree view - optional" on page 1178
- "Navigate within the Data - optional" on page 1178
- "Create and navigate with bookmarks - optional" on page 1179

- "Insert data into a buffer - optional" on page 1180
- "Copy and paste blocks of data - optional" on page 1180

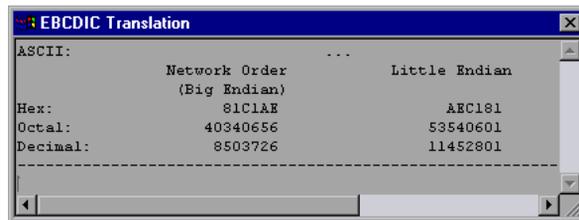
View and modify the data in Script view - optional

In Script view, select the **data.ws** file in the left pane. If necessary, modify the data directly in the VuGen editor window. For details, see "Windows Socket Data" on page 1164.

Open the buffer in a viewer- optional

To display the data in its EBCDIC translated form, select it in Script view 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.



- If the selected text is more than four characters, VuGen displays the data in several columns in **long format**.

For details, see the "Data Viewer" on page 1168.

View and modify the data in Tree view - optional



In Tree view, the right pane displays a snapshot of the data. To edit the data, clear the **Read only** option in the buffer's **Text** view. VuGen issues a warning that bookmarks and parameters may be affected. For details, see "WinSock Data Buffer Snapshot" on page 1166.

Navigate within the Data - optional

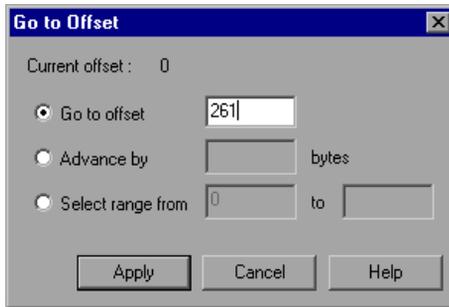
To navigate within the buffer data, use the **Buffer Navigator** or the **Go to Offset** dialog box.

Buffer Navigator

Enter Tree view. Select **View > Buffer Navigator**. If desired, select a filter, for example Receive buffers. Select the buffer's whose data you want to view. If necessary, modify the data in the snapshot's Text view. For details, see "Buffer Navigation" on page 1170.

Go to Offset

Click in the Snapshot window. Select **Go to Offset** from the shortcut menu.



- To go to a specific offset within the buffer (absolute), select **Go to offset** and specify an offset value. Click **Apply**.
- To jump to a location relative to the cursor, click **Advance by** and specify the number of bytes to advance. To advance ahead, enter a positive value. To move backwards within the buffer, use a negative value. Click **Apply**.

- To select a range of data within the buffer, click **Select range from** and specify the beginning and end offsets. Click **Apply**.

For details, see "Jump to Location" on page 1171.

Create and navigate with bookmarks - optional

Bookmarks let you remember certain locations within the data buffers. For details, see "Bookmarking" on page 1171.

- 1** To create a bookmark, open Tree view and select one or more bytes in a buffer snapshot (Text or Binary view). Select **New Bookmark** from the shortcut 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 from being Permanent to simple (permanent means that it is always marked, even when you move the cursor to a new location), select the bookmark, and clear the check adjacent to **Permanent Bookmark** in the shortcut menu.
- 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**.

Insert data into a buffer - optional

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 snapshot (Text or Binary view).
- 2 Select **Advanced > Insert Number > Specify...** from the shortcut menu.
- 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.

Copy and paste blocks of data - optional

You can modify the data as characters, decimal numbers, or hexadecimal numbers. For details, see "Buffer Data Editing" on page 1172.



- 1 Open the snapshot's Binary view .
- 2 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 shortcut menu.
 - As a hexadecimal number, **Advanced > Copy As Number > Hexadecimal** in the shortcut menu.
- 3 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 shortcut menu.
 - In long format (4-byte), **Advanced >Insert Number >Paste Long (4-byte)** in the shortcut menu.

- 4 To delete data, select it in the Text or Binary views and select **Delete** from the shortcut menu.

Reference

Data Buffers

The **data.ws** data file has the following format:

- ▶ File header
- ▶ A list of buffers and their contents

The file header includes an internal version number of the data file format. The current version is 2. If you try to access data from a data file with format version 1, VuGen issues an error.

```
;WSRData 2 1
```

An identifier precedes each record, indicating whether the data was received or sent, followed by the buffer descriptor, and the number of bytes received (for **lrs_receive** only). The buffer descriptor contains a number identifying the buffer.

For example,

```
recv buf5 25
```

indicates that the buffer contains data that was received. The record number is 5, indicating that this receive operation was the sixth data transfer (the index is zero based), and twenty-five bytes of data were received.

If your data is in ASCII format, the descriptor is followed by the actual ASCII data that was transferred by the sockets.

If your data is in EBCDIC format, it must be translated through a look-up table. For information on setting the translation table, see the "WinSocket Node" on page 413. 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"
```

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:

<code>\a</code>	Bell (alert)
<code>\b</code>	Backspace
<code>\f</code>	Formfeed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\'</code>	Single quotation mark
<code>\"</code>	Double quotation mark
<code>\\</code>	Backslash
<code>\?</code>	Literal question mark
<code>\ooo</code>	ASCII character -octal

The supported conversion characters are:

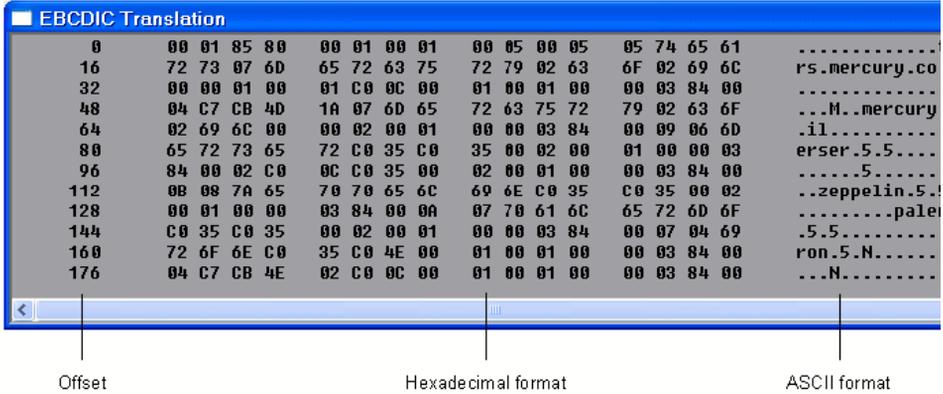
%a	ASCII representation
%BX	Big Endian (Network Order) Hex
%BO	Big Endian (Network Order) Octal
%BD	Big Endian (Network Order) Decimal
%LX	Little Endian Hex
%LO	Little Endian Octal
%LD	Little Endian Decimal

- **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.

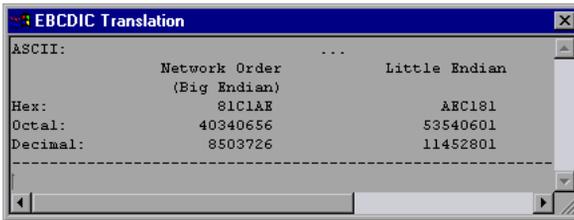
In the default settings, long and short formats are set, and a dot is specified for non-printable characters.

```
[BufferFormats]
LongBufferFormat=nnnnnnnn  XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX
XX XX  aaaaaaaaaaaaaaaaaa\r\n
LongBufferHeader=
LongBufferFooter=
ShortBufferFormat=ASCII:\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\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:



Windows Socket User Interface

This section includes (in alphabetical order):

- ▶ Data Navigator Dialog Box on page 1187
- ▶ Go To Offset Dialog Box on page 1187
- ▶ Bookmarks Dialog Box on page 1188

Data Navigator Dialog Box

This dialog box displays the send and receive buffer steps. When you select a buffer in the navigator, its contents are displayed in the buffer snapshot window.

To access	View > Buffer Navigator
Important information	You can also navigate between buffers by clicking on the buffer step in the tree view. The advantages of the Buffer Navigator are that it is a floating window with filtering capabilities.
Relevant tasks	"Navigate within the Data - optional" on page 1178

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

UI Elements (A-Z)	Description
<buffer list>	A list of all the Send and Receive buffers generated during recording.
Filter	Filters the data displayed in the buffer snapshot window. The drop-down list provides these options: <ul style="list-style-type: none"> ▶ All buffers ▶ Receive buffer ▶ Send buffer

Go To Offset Dialog Box

This dialog box allows you to go to a specific location within the recorded data.

To access	Click within the Snapshot window and choose Go to offset from the shortcut menu.
Relevant tasks	"Navigate within the Data - optional" on page 1178

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

UI Elements (A-Z)	Description
	Moves the cursor to the specified offset.
Current offset	The current offset of the cursor (read only).
Go to offset	Goes to a specific, absolute offset within the data.
Advance by...bytes	Jumps to a location relative to the cursor, by a number of bytes. Positive values indicate a forward direction. Negative values indicate a reverse direction.
Select range from...to...	Selects a range of data within the buffer.

Bookmarks Dialog Box

This dialog box allows you to set bookmarks and navigate to them.

To access	To view the bookmark list, select View > Output Window and select the Bookmarks tab.
Relevant tasks	"Create and navigate with bookmarks - optional" on page 1179

UI Elements (A-Z)	Description
	Goes to the selected bookmark.
	Creates a new bookmark at the current location of the cursor.
	Deletes the selected bookmark.
	Changes the selected bookmark to the current location of the cursor in the data buffer.

UI Elements (A-Z)	Description
<bookmark list>	<p>A list of all the bookmarks with their information:</p> <ul style="list-style-type: none"> ▶ Name. The name given to the bookmark. ▶ Action. The script section containing the bookmark. ▶ Buffer. The bookmark's buffer name. ▶ Location. The range of the bookmark in the buffer. ▶ Description. The type of bookmark: Permanent or Simple. <p>Tips:</p> <ul style="list-style-type: none"> ▶ Double-click an entry to move the cursor to the bookmarked data. ▶ Click a column to sort by that information.
Filter	<p>Filters the bookmark list by:</p> <ul style="list-style-type: none"> ▶ All buffers ▶ Receive buffers ▶ Send buffers ▶ Selected buffer only
Show permanent bookmarks only	<p>Displays only Permanent bookmarks in the list—not Simple ones.</p>

41

Wireless Protocols

This chapter includes:

Concepts

- WAP Protocol Overview on page 1192
- WAP Toolkits on page 1193
- Push and Pull Technology on page 1194
- VuGen Push Support on page 1195
- MMS (Multimedia Messaging Service) Protocol Overview on page 1197

Tasks

- How to Run an MMS Scenario in the Controller on page 1198

Concepts

WAP Protocol Overview

The Wireless Application Protocol (WAP) is an open, global specification that enables mobile users with wireless devices to instantly access and interact with information and services.

The WAP protocol specifies a microbrowser thin-client using a new standard called WML that is optimized for wireless handheld mobile terminals. WML is a stripped-down version of XML.

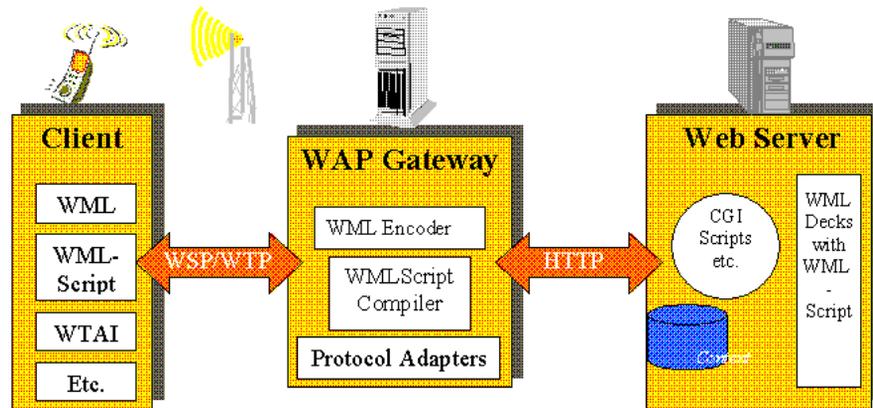
WAP also specifies a proxy server that:

- ▶ acts as a gateway between the wireless network and the wire-line Internet
- ▶ provides protocol translation
- ▶ optimizes data transfer for the wireless handset

WAP architecture closely resembles the WWW model. All content is specified in formats that are similar to the standard Internet formats. Content is transported using standard protocols in the WWW domain and an optimized HTTP-like protocol in the wireless domain (Wireless Session Protocol). You locate all WAP content using WWW standard URLs.

WAP uses many WWW standards, including authoring and publishing methods. WAP enhances some of the WWW standards in ways that reflect the device and network characteristics. WAP extensions are added to support Mobile Network Services such as Call Control and Messaging. It accounts for the memory and CPU processing constraints that are found in mobile terminals. WAP also supports low bandwidth and high latency networks.

WAP assumes the existence of a gateway that is responsible for encoding and decoding data transferred to and from the mobile client. The purpose of encoding content delivered to the client is to minimize the size of data sent to the client over-the-air, as well as to minimize the computational energy required by the client to process that data. The gateway functionality can be added to origin servers, or placed in dedicated gateways as illustrated below.



WAP Toolkits

To assist developers in producing WAP applications and services, the leading companies such as Nokia, Ericsson, and Phone.com, have developed toolkits. The WAP Toolkit provides an environment for developers who want to provide Internet services and content for mobile terminals. It allows developers to write, test, debug, and run applications on a PC-based simulator phone. The toolkit allows users to browse WAP sites through an HTTP connection or a WAP gateway.

A mobile phone communicates with a gateway in WSP protocol; a toolkit can communicate with the gateway, or directly with the server. VuGen automatically detects the communication mode that is configured in the toolkit: WSP or HTTP. If you are interested in the traffic to the gateway, you record in WSP mode. If you want to check the server and the content providers, you can record your toolkit session in HTTP mode, and bypass the gateway.

VuGen uses custom API functions to emulate a user session. Most functions are the standard Web protocol functions utilizing the HTTP protocol. Several WAP functions emulate actions specific to WAP Vusers.

Push and Pull Technology

In the normal client/server model, a client requests information or a service from a server. The server responds by transmitting information or performing a service to the client. This is known as **pull** technology—the client pulls information from the server.

In contrast to this, there is also **push** technology. The WAP push framework transmits information to a device without a previous user action. This technology is also based on the client/server model, but there is no explicit request from the client before the server transmits its content.

To perform a push operation in WAP, a **Push Initiator** (PI) transmits content to a client. However, the Push Initiator protocol is not fully compatible with the WAP Client—the Push Initiator is on the Internet, and the WAP Client is in the WAP domain. Therefore, we need to insert a translating gateway to serve as an intermediary between the Push Initiator and the WAP Client. The translating gateway is known as the **Push Proxy Gateway** (PPG).

The access protocol on the Internet side is called the **Push Access Protocol** (PAP).

The protocol on the WAP end is called the Push **Over-The-Air** (OTA) protocol.

The Push Initiator contacts the Push Proxy Gateway (PPG) over the Internet using the PAP Internet protocol. PAP uses XML messages that may be tunneled through various well-known Internet protocols such as HTTP. The PPG forwards the pushed content to the WAP domain. The content is then transmitted using the OTA protocol over the mobile network to the destination client. The OTA protocol is based on WSP services.

In addition to providing basic proxy gateway services, the PPG is capable of notifying the Push Initiator about the final status of the push operation. In two-way mobile networks, it can also wait for the client to accept or reject the content.

Push Services Types

Push services can be of the SL or SI type:

- ▶ **SL.** The Service Loading (SL) content type provides the ability to cause a user agent on a mobile client to load and execute a service—for example, a WML deck. The SL contains a URI indicating the service to be loaded by the user agent without user intervention when appropriate.
- ▶ **SI.** The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. For example, the notifications may be about new emails, changes in stock price, news headlines, and advertising.

In its most basic form, an SI contains a short message and a URI indicating a service. The message is presented to the end-user upon reception, and the user is given the choice to either start the service indicated by the URI immediately, or postpone the SI for later handling. If the SI is postponed, the client stores it and the end-user is given the ability to act upon it at a later point of time.

VuGen Push Support

Push support for VuGen is divided into three parts:

- ▶ Push support at the client end—the ability to accept push messages.
- ▶ Push support to WAP HTTP Vusers—emulating Push Initiators.
- ▶ Push messages (SI & SL) format services—formatting push messages.

Client Push Support

At the client end, VuGen supports both push services (SL and SI) for all replay modes (CO and CL). The `wap_wait_for_push` function instructs the Vuser to wait for a push message to arrive. You set the timeout for this function in the run-time settings.

When a push message arrives, the Vuser parses it to determine its type and to retrieve its attributes. If parsing was successful, it generates and executes a pull transaction to retrieve the relevant data. You can disable the pull event, indicating to the Vuser not to retrieve the message data by configuring the Run-Time settings.

Emulating Push Initiators

Push support for WAP HTTP Vusers enables you to perform load testing of the PPG. Push support allows Vusers to function as Push Initiators supporting the **Push Access Protocol** (PAP). The PAP defines the following sets of operations between the PI and the PPG:

- ▶ Submit a Push request.
- ▶ Cancel a Push request.
- ▶ Submit a query for the status of a push request.
- ▶ Submit a query for the status of a wireless device's capabilities.
- ▶ Initiate a result notification message from the PPG to the PI.

All operations are request/response—for every initiated message, a response is issued back to the PI. PI operations are based on the regular HTTP POST method supported by VuGen. Currently, only the first two operations are supported through `wap_push_submit` and `wap_push_cancel`.

Formatting Push Messages

You can submit data to a Web server using the `web_submit_data` function. It is difficult, however, to send long and complex data structures using this function. To overcome this difficulty and provide a more intuitive API function, several new API functions were added to properly format the XML message data: `wap_format_si_msg` and `wap_format_sl_msg`. For more information about these functions, see the *Online Function Reference*.

MMS (Multimedia Messaging Service) Protocol Overview

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**).

Tasks

How to Run an MMS Scenario in the Controller

An MMS (Multimedia Messaging Service) scenario requires a command line setting.

To set the MMS command line setting:

- 1** From the Scenario Schedule screen, click **Details**. The Group Information dialog is displayed.
- 2** If the Command line box is not visible, click the **More** button.
- 3** Add the following to the end of the Command line text: `-usingwininet yes`
- 4** Click **OK** to accept the Command line switch.

Part III

Advanced Topics



42

Manually Programming a Script using the VuGen Editor

This chapter includes:

Concepts

- Manually Programming Scripts - Overview on page 1202
- C Vuser Scripts on page 1203
- JavaScript Vusers on page 1205
- VBScript Vusers on page 1206
- Java Vusers on page 1207
- VB Vusers on page 1208

Concepts

Manually Programming Scripts - Overview

VuGen allows you to program your own functions into the script, instead of recording an actual session. You can use the Vuser API or standard programming functions. Vuser API functions allow you to gather information about Vusers. For example, you can use Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test or monitoring.

This chapter describes how to program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes.

You can also develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API function libraries. For more information, see "Creating Scripts with Visual Studio" on page 1211.

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 Vuser Scripts

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.

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. */
```

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, the following code creates the standard object **lr**:

```
var lr = new ActiveXObject("LoadRunner.LrApi")
```

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, the following code creates the standard object **lr**:

```
Set lr = CreateObject("LoadRunner.LrApi")
```

 **Java Vusers**

In Java Vuser scripts, you can place any standard Java code. To create an empty Java Vuser script, select Java Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty Java script:

```
import Irapl.Ir;  
  
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.

43

Creating Scripts with Visual Studio

This chapter includes:

Concepts

- ▶ Creating Vuser Scripts in Visual Studio - Overview on page 1212

Tasks

- ▶ How to Create a Vuser Script with Visual C on page 1214
- ▶ How to Create a Vuser Script with Visual Basic on page 1215
- ▶ How to Configure Run-Time Settings and Parameters on page 1217

Concepts

Creating Vuser Scripts in Visual Studio - Overview

There are several ways to create Vuser scripts: through VuGen or a development environment such as Visual Studio.

<i>VuGen</i>	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.
<i>Visual Studio</i>	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 42, "Manually Programming a Script using 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

An online C reference of the common functions used in Vuser scripts, are included in the *Online Function Reference* (**Help > Function Reference**).

Tasks

How to Create a Vuser Script with Visual C

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 **lruser.usr** file in the *Template* directory, Update the USR file key *BinVuser* with the DLL name: `BinVuser=<DLL_name>`.

Example:

In the following example, the `lr_output_message` 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, `lrn.h` is correct.

```
#include "c:\lrn_5\include\lrn.h"

extern "C" {
int __declspec(dllexport) Init (void *p)
{
    lr_output_message("in init");
return 0;
}

int __declspec(dllexport) Run (void *p)
{
    const char *str = lr_eval_string("<name>");
    lr_output_message("in run and parameter is %s", str);
return 0;
}

int __declspec(dllexport) End (void *p)
{
    lr_output_message("in end");
return 0;
}
} //extern C end
```

How to Create a Vuser Script with Visual Basic

To create a Vuser in Visual Basic:

- 1** In Microsoft Visual Basic, create a new project by selecting **File > New Project > LoadRunner Virtual User**. A new project is created with one class and a template for a Vuser.
- 2** Save the project by selecting **File > Save Project**.

- 3 Open the Object Browser (View menu). Select the LoadRunner Vuser library and double-click on the Vuser Class module to open the template. The template contains three sections, **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
```

- 4 Add code from your client application, or program as you normally would.
- 5 Use the Object Browser to add the desired VuGen elements to your code such as transactions, think time, rendezvous points, and messages.
- 6 Enhance your program with run-time settings and parameters. For more information, see "How to Configure Run-Time Settings and Parameters" on page 1217.
- 7 Build the Vuser script by selecting **File > Make *project_name*.dll**.
The project is saved in the form of a Vuser script (.usr) in the same directory as the project.

How to Configure Run-Time Settings and Parameters

After you create the DLL for your script, you create a script (*.usr*) and configure its settings. The *lrbin.bat* utility provided with VuGen lets you define parameters and configure runtime settings for scripts created with Visual C and Basic. This utility is located in the *bin* 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 "Run-Time Settings" on page 417
- 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 "Parameters" on page 257

Test the script by running it in standalone mode. Select **Vuser > Run Vuser**. The Vuser execution window appears while the script runs.

44

Language Support

This chapter includes:

Concepts

- ▶ Language Support - Overview on page 1220
- ▶ Page Request Header Language on page 1220

Tasks

- ▶ How to Convert Encoding Format of a String on page 1221
- ▶ How to Convert Encoding Format of Parameter Files on page 1222
- ▶ How to Record Web Pages with Foreign Languages on page 1224

Reference

Troubleshooting and Limitations on page 1226

Concepts

Language Support - Overview

VuGen supports multilingual environments, allowing you to use languages other than English on native language machines when creating and running scripts.

When working with languages other than English, the primary issue is ensuring that VuGen recognizes the encoding of the text during record and replay. The encoding applies to all texts used by the script. This includes texts in HTTP headers and HTML pages for Web Vusers, data in parameter files, and others.

Windows 2000 and higher lets you save text files with a specific encoding directly from Notepad: ANSI, Unicode, Unicode big endian, or UTF-8.

By default, VuGen works with the local machine encoding (ANSI). Some servers working with foreign languages, require you to work with UTF-8 encoding. To work against this server, you must indicate in the Advanced recording options, that your script requires UTF-8 encoding.

Page Request Header Language

Before running a Web script, you can set the page's request header to match your current language. In the Internet Protocol Run-Time settings, you set the language of the *Accept-Language* request header. This header provides the server with a list of all of the accepted languages.

To set this value, select **Vuser > Run-Time Settings > Internet Protocol > Preferences > Advanced > Options > Accept-Language request header** and select the desired language.

For user interface details, see "Internet Protocol Preferences Node" on page 456.

Tasks

How to Convert Encoding Format of a String

You can manually convert a string from one encoding to another (UTF-8, Unicode, or locale machine encoding) using the `lr_convert_string_encoding` function with the following syntax:

```
lr_convert_string_encoding(char * sourceString, char * fromEncoding, char *
toEncoding, char * paramName)
```

The function saves the result string (including its terminating NULL) in the third argument, *paramName*. It returns a 0 on success and -1 on failure.

The format for the **fromEncoding** and **toEncoding** arguments are:

LR_ENC_SYSTEM_LOCALE	NULL
LR_ENC_UTF8	"utf-8"
LR_ENC_UNICODE	"ucs-2"

In the following example, `lr_convert_string_encoding` converts "Hello world" from the system locale to Unicode.

```
Action()
{
    int rc = 0;
    unsigned long converted_buffer_size_unicode = 0;
    char          *converted_buffer_unicode = NULL;

    rc = lr_convert_string_encoding("Hello world", NULL, LR_ENC_UNICODE,
"stringInUnicode");
    if(rc < 0)
    {
        // error
    }
    return 0;
}
```

In the replay log, the output window shows the following information:

```
Output:
Starting action Action.
Action.c(7): Notify: Saving Parameter "stringInUnicode = H\x00e\x00l\x00l\x00o\x00
\x00w\x00o\x00r\x00l\x00d\x00l\x00\x00"
Ending action Action.
```

The result of the conversion is saved to the *paramName* argument.

How to Convert Encoding Format of Parameter Files

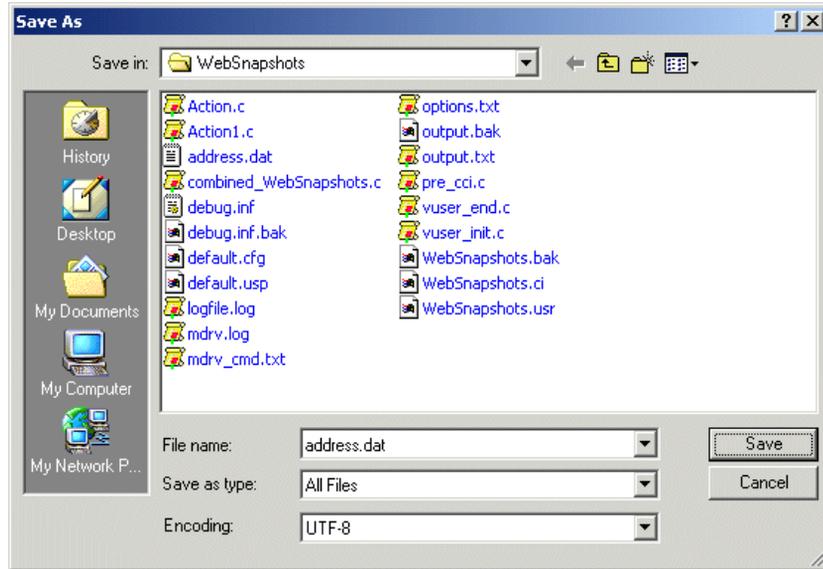
The parameter file contains the data for parameters that were defined in the script. This file, stored in the script's directory, has a *.dat* extension. When running a script, Vusers use the data to execute actions with varying values.

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.

How to Record Web Pages with Foreign Languages

When working with Web or other Internet protocols, you can indicate the encoding of the Web page text for recording. The recorded site's language must match the operating system language. You cannot mix encodings in a single recording—for example, UTF-8 together with ISO-8859-1 or shift_jis.

This task describes how to record web pages with foreign languages using VuGen.

This task includes the following steps:

- ▶ "Automatically record foreign language web pages." on page 1224
- ▶ "Manually record foreign language web pages" on page 1225

1 Automatically record foreign language web pages.

In order to be recognized as a non-English Web page, the page must indicate the charset in the HTTP header or in the HTML meta tag. Otherwise, VuGen will not detect the EUC-JP encoding and the Web site will not be recorded properly. To instruct VuGen to record non-English requests as **EUC-JP** or **UT-8**, select **Tools > Recording Options > HTTP Properties > Advanced > support charset** and select the appropriate option in the Recording Options dialog box, **HTTP Properties: Advanced** node. For user interface details, see "HTTP Advanced Node" on page 362.

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.

2 Manually record foreign language web pages

You can manually add full support for recording and replaying of HTML pages encoded in EUC-JP using the `web_sjis_to_euc_param` function. This also allows VuGen to display Japanese EUC-encoded characters correctly in Vuser scripts.

When you use `web_sjis_to_euc_param`, VuGen shows the value of the parameter in the Execution Log using EUC-JP encoding. For example, when you replay the `web_find` function, VuGen displays the encoded values. These include string values that were converted into EUC by the `web_sjis_to_euc_param` function, or parameter substitution when enabled in the **Run-Time Setting > Log > Extended Log**.

Reference

Troubleshooting and Limitations

This section describes troubleshooting and limitations when working with foreign languages.

Browser Configuration

If, during recording, non-English characters in the script are displayed as escaped hexadecimal numbers (For example, the string "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*.

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).

Application Lifecycle Management Integration

To open a script saved in an Application Lifecycle Management project from VuGen, or a scenario saved in an ALM project from Controller, add a new Test Set named "Default" (in English) to the ALM project.

45

Advanced Topics

This chapter includes:

Concepts

- ▶ Calling External Functions in DLL's on page 1230
- ▶ Recording OLE Servers on page 1230
- ▶ Running a Vuser from the Unix Command Line on page 1233
- ▶ Specifying the Vuser Behavior on page 1234
- ▶ Command Line Parameters on page 1235

Tasks

- ▶ How to Create a New Vuser Type on page 1236
- ▶ How to Load a DLL locally on page 1241
- ▶ How to Load a DLL Globally on page 1242

Reference

- ▶ .dat Files on page 1244

Concepts

Calling External Functions in Dll's

You can call functions that are defined in external DLLs. By calling external functions from your script, you can reduce the memory footprint of your script and the overall run-time.

To call the external function, you load the DLL in which the function is defined.

You can load a DLL in one of the following ways:

- ▶ Locally (for one script) by using the `lr_load_dll` function. For task details, see "How to Load a DLL locally" on page 1241.
- ▶ Globally (for all scripts) by adding statements to the **vugen.dat** file. For task details, see "How to Load a DLL Globally" on page 1242.

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 1232 for a list of the CmdLine environment variables.
- 3** Start the application (not from VuGen). Notepad opens with the full command line. Check the command line arguments. Start the application several times and compare the command line arguments. If the arguments are the same each time you invoke the application, then you can reset the CmdStartNotepad environment variable. Otherwise, leave it set to "1".
- 4** In VuGen, invoke the program, MyOleSrv.1.exe with the command line parameters (use Copy/Paste from the Notepad window).
- 5** Start the application (not from within VuGen).

CmdLine Environment Variables

You can control the execution of CmdLine through the following environment variables:

- | | |
|------------------------|---|
| CmdNoPopup | If set, the popup window will not appear. |
| CmdOutFileName | If set, and non-empty, CmdLine will attempt to create this file instead of c:\temp\CmdLine.txt. |
| CmdStartNotepad | If set, the output file will be displayed in the notepad (Best used with CmdNoPopup). |

Running a Vuser from the 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 preprocessing phase. The output of this process is the file '*Vuser.c*'.
- cci_only* This option runs the compilation phase. The '*Vuser.c*' file is used as input, and the output produced is the '*Vuser.ci*' file.
- exec_only* This option runs the Vuser, by taking as input the '*Vuser.ci*' file and running it via the replay driver.
- ci ci_file* This option allows you to specify the name and location of a *.ci* file to be run. The second parameter contains the location of the *.ci* file.
- out output_directory* This option allows you to determine the location of any output files created throughout the various processes. The second parameter is the directory name and location.
- driver driver_path* This option allows you to specify the actual driver executable to be used for running the Vuser. By default the driver executable is taken from the settings in the VuGen.dat file.

Note that only one of the first three options can be used at a time for running the *run_db_vuser*.

Specifying the Vuser Behavior

Since VuGen creates the Vuser script and the Vuser behavior as two independent sources, you can configure user behavior without directly referencing the Vuser script, for example, wait times, pacing times, looping iterations, logging, and so forth. This feature lets you make configuration changes to a Vuser, as well as store several 'profiles' for the same Vuser script.

The '*Vuser.cfg*' file, by default, is responsible for defining this behavior - as specified in VuGen's Runtime settings dialog box. You can save several versions of this file for different user behavior and then run the Vuser script referencing the relevant *.cfg* file.

You can run the Vuser script with the relevant configuration file from a server machine. To do this, add the following to the Vuser command line:

```
-cfg c:\tmp\profile2.cfg
```

For information on command line parameters, see "Command Line Parameters" on page 1235.

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.

Tasks

How to Create a New Vuser Type

The following steps describe how to create a new vuser type.

- "Edit the mdrv.dat file" on page 1237
- "Add a CFG file (optional)" on page 1239
- "Insert an LRP file" on page 1240
- "Specify a Template" on page 1241

1 Edit the `mdrv.dat` file

Edit the `mdrv.dat` file which resides in the `M_LROOT\dat` directory. Add a section for the new `Vuser` type with all of the applicable parameters from the following list.

```
[<extension_name>]
ExtPriorityType=< {internal, protocol}>
WINNT_EXT_LIBS=<dll name for NT>
WIN95_EXT_LIBS=<dll name for 95>
SOLARIS_EXT_LIBS=<dll name for Solaris>
LINUX_EXT_LIBS=<dll name for Linux>
HPUX_EXT_LIBS=<dll name for HP>
AIX_EXT_LIBS=<dll name for IBM>
LibCfgFunc=<configuration function name>
UtilityExt=<other extensions list>
WINNT_DLLS=<dlls to load to the interpreter context, for NT>
WIN95_DLLS=<dlls to load to the interpreter context, for 95>
SOLARIS_DLLS=<dlls to load to the interpreter context, for Solaris>
LINUX_DLLS=<dlls to load to the interpreter context, for Linux>
HPUX_DLLS=<dlls to load to the interpreter context, for HP>
AIX_DLLS=<dlls to load to the interpreter context, for IBM>
ExtIncludeFiles=<extra include files. several files can be seperated by a comma>
ExtCmdLineConc=<extra command line (if the attr exists concatenate value)>
ExtCmdLineOverwrite=<extra command line (if the attr exists overwrite value)>
CallActionByNameFunc=<interpreter exec_action function>
GetFuncAddress=<interpreter get_location function>
RunLogicInitFunc=<action_logic init function>
RunLogicRunFunc=<action_logic run function>
RunLogicEndFunc=<action_logic end function>
```

For example, an Oracle NCA Vuser type is represented by:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp11i.dll
WIN95_EXT_LIBS=ncarp11i.dll
LINUX_EXT_LIBS=liboranca11i.so
SOLARIS_EXT_LIBS=liboranca11i.so
HPUX_EXT_LIBS=liboranca11i.sl
AIX_EXT_LIBS=liboranca11i.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api,HttpEngine
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
SecurityRequirementsFiles=oracle_nca.asl
SecurityMode=On
```

VuGen was designed to be able to handle a new Vuser type with no code modifications. You may, however, need to add a special View.

There is no generic driver supplied with VuGen, but you can customize one of the existing drivers. To use a customized driver, modify *mdrv.dat*. Add a line with the platform and existing driver, then add a new line with your customized driver name, in the format `<platform>_DLLS=<my_replay.dll name>`. For example, if your SAP replay dll is called SAPPLAY32.DLL, add the following two lines to the [sap] section of *mdrv.dat*:

```
WINNT=sapdrv32.exe
WINNT_DLLS=sapplay32.dll
```

2 Add a CFG file (optional)

You can specify a configuration file to set the default Run-Time Settings for your protocol. You define it in the LibCfgFunc variable in the mdrv.dat file, or place one called default.cfg in the new protocols 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
```

3 Insert 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.

4 Specify 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"
```

How to Load a DLL locally

This task describes how to use the **lr_load_dll** function to load a DLL into your Vuser script. Once the DLL is loaded, you can call any function defined within the DLL without having to declare it in your script.

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.

How to Load a DLL Globally

This task describes how to load a DLL globally, to make its functions available to all your Vuser scripts. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

To load a DLL globally:

- 1 Add a list of the DLLs you want to load to the appropriate section of the `mdrv.dat` file, located in your application's `dat` 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=wrun32.dll
WIN95_EXT_LIBS=wrun32.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.

Reference

.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. For information about how to use this file to create a new protocol, see "How to Create a New Vuser Type" on page 1236.

46

Creating and Running Script in UNIX

This chapter includes:

Concepts

- ▶ Creating and Running Scripts in UNIX - Overview on page 1248
- ▶ Programming Vuser Actions on page 1248

Tasks

- ▶ How to Create a Template on page 1251
- ▶ How to Configure Run-Time Settings Manually on page 1252
- ▶ How to Define Transaction and Insert Rendezvous Points Manually on page 1257
- ▶ How to Compile Scripts Manually on page 1257

Concepts

Creating and Running Scripts in UNIX - Overview

You can use VuGen on a UNIX environment in the following ways:

- ▶ 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.
- ▶ 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.

To create a script through programming, you can use a Vuser template as a basis for a larger Vuser scripts. The template provides:

- ▶ correct program structure
- ▶ Vuser API calls
- ▶ source code and makefiles for creating a dynamic library

Programming Vuser Actions

The Vuser script files, *test.c*, *test.usr*, and *test.cfg*, can be customized for your Vuser.

You program the actual Vuser actions into the *test.c* file. This file has the required structure for a programmed Vuser script. The Vuser script contains three sections: *vuser_init*, *Actions*, and *vuser_end*.

Note that the template defines extern C for users of C++. This definition is required for all C++ users, to make sure that none of the exported functions are modified inadvertently.

```
#include "lrun.h"
#if defined(__cplusplus) || defined(cplusplus) extern "C"
{
#endif
int LR_FUNC vuser_init(LR_PARAM p)
{
    lr_message("vuser_init done\n");
    return 0;
}
int Actions(LR_PARAM p)
{
    lr_message("Actions done\n");
    return 0;
}
int vuser_end(LR_PARAM p)
{
    lr_message("vuser_end done\n");
    return 0;
}
#endif
#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.

Tasks

How to Create a Template

VuGen includes a utility that copies a template into your working directory. The utility is called `mkbdbtest`, and is located in `$M_LROOT/bin`. You run the utility by typing:

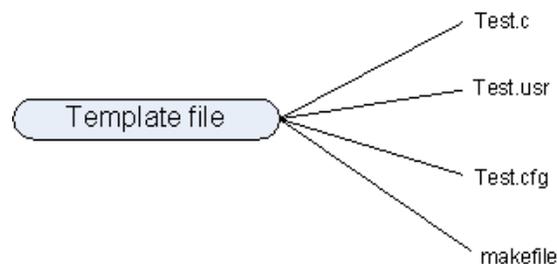
```
mkbdbtest name
```

When you run `mkbdbtest`, it creates a directory called `name`, which contains the template file, `name.c`. For example, if you type:

```
mkbdbtest test1
```

`mkbdbtest` creates a directory called `test1`, which contains the template script, `test1.c`.

When you run the `mkbdbtest` 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 `mkbdbtest`.



How to Configure Run-Time Settings Manually

To configure Vuser run-time settings, you modify the *default.cfg* and *default.usp* files created with the script. These run-time settings correspond to VuGen's run-time settings. (See Chapter 11, "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 option 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

Option	Options	Factor	LimitFlag	Limit
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]
MerClniTreeFather=""
MerClniTreeSectionName="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"
MerClniTreeSons="Action,Action2,Action3"
RunLogicPaceConstAfterTime="30.000"
```

How to Define Transaction and Insert Rendezvous Points Manually

When programming a Vuser script without VuGen, you must manually configure the Vuser file in order to enable transactions and rendezvous. The configuration settings are listed in the test.usr file.

```
[General]
Type=any
DefaultCfg=Test.cfg
BinVuser=libtest.libsuffix
RunType=Binary

[Actions]
vuser_init=
Actions=
vuser_end=

[Transactions]
transaction1=

[Rendezvous]
Meeting=
```

Each transaction and rendezvous must be defined in the *usr* file. Add the transaction name to the Transactions section (followed by an "="). Add each rendezvous name to the Rendezvous section (followed by an "="). If the sections are not present, add them to the *usr* file as shown above.

How to Compile Scripts Manually

After you modify the template, you compile it with the appropriate *Makefile* in the script's 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 \  
        -lrun50 \  
        -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 configuration. For more information, see the *HP LoadRunner Controller, Performance Center, or HP Business Service Management* documentation.

47

Programming with the XML API

This chapter includes:

Concepts

- ▶ Programming with the XML API - Overview on page 1260
- ▶ Using XML Functions on page 1261
- ▶ Specifying XML Function Parameters on page 1264
- ▶ XML Attributes on page 1266
- ▶ Structuring XML Scripts on page 1266
- ▶ Enhancing a Recorded Session with XML on page 1268

Tasks

- ▶ How to Use Result Parameters on page 1273

Concepts

Programming with the XML API - Overview

VuGen's support for XML allows you to dynamically work with XML code and retrieve the values during test execution. Follow these steps in creating an effective XML script:

- ▶ Record a script in the desired protocol, usually Web, Web Services, or Wireless.
- ▶ Copy the XML structures into your script.
- ▶ Add XML functions from the LR API in order to retrieve dynamic data and the XML element values.

The LR API uses XPath, the XML Path language to manipulate the text in an XML document.

You can instruct VuGen to display the output values of XML elements in the Execution log window using the Run-Time settings. VuGen displays the line numbers, the number of matches, and the value. To allow the displaying of values, you need to enable parameter substitution. In the Run-Time settings, open the **General:Log** node, select **Extended log**, and select **Parameter Substitution**. For more information, see Chapter 11, "Run-Time Settings."

All Vuser API XML functions return the number of matches successfully found, or zero for failure.

Using XML Functions

This section provides examples of how to work with data in an XML tree. Certain functions allow you to retrieve information, and others let you write information to an XML tree. The examples use the following XML tree containing the names and extensions of several employees in the Acme organization.

```
<acme_org>
  <accounting_dept>
    <employee type='PT'>
      <name>John Smith</name>
      <extension>2145</extension>
    </employee>
  </accounting_dept>
  <engineering_dept>
    <employee type='PT'>
      <name>Sue Jones</name>
      <extension>2375</extension>
    </employee>
  </engineering_dept>
</acme_org>
```

Reading Information from an XML Tree

The functions which read information from an XML tree are:

lr_xml_extract	Extracts XML string fragments from an XML string.
lr_xml_find	Performs a query on an XML string.
lr_xml_get_values	Retrieves values of XML elements found by a query.

To retrieve a specific value through a query, you specify the tags of the parent and child nodes in a path format.

For example, to retrieve an employee name in the Accounting department, use the following string:

```
lr_xml_get_values("XML={XML_Input_Param}",
"ValueParam=OutputParam",
"Query=/acme_org/accounting_dept/employee/name",
LAST);
```

The Execution log window (with Extended logging enabled) shows the output of this function:

Output:

Action.c(20): "lr_xml_get_values" was successful, 1 match processed
Action.c(25): Query result = **John Smith**

Writing to an XML Structure

The functions which write values to an XML tree are:

lr_xml_delete	Deletes fragments from an XML string.
lr_xml_insert	Inserts a new XML fragment into an XML string.
lr_xml_replace	Replaces fragments of an XML string.
lr_xml_set_values	Sets the values of XML elements found by a query.
lr_xml_transform	Applies Extensible Stylesheet Language (XSL) transformation to XML data.

The most common *writing* function is **lr_xml_set_values** which sets the values of specified elements in an XML string. The following example uses **lr_xml_set_values** to change the phone extensions of two *employee* elements in an XML string.

First, we save the XML string to a parameter called *XML_Input_Param*. We want two values to be matched and substituted, so we prepare two new parameters, *ExtensionParam_1* and *ExtensionParam_2*, and set their values to two new phone extensions, 1111 and 2222.

`lr_xml_set_values` contains the argument "ValueName=ExtensionParam", which picks up the values of `ExtensionParam_1` and `ExtensionParam_2`. The current extensions of the two employees are substituted with the values of these parameters, 1111 and 2222. The value of `OutputParam` is then evaluated proving that the new phone extensions were in fact substituted.

```

Action() {
    int i, NumOfValues;
    char buf[64];

    lr_save_string(xml_input, "XML_Input_Param"); // Save input as parameter
    lr_save_string("1111", "ExtensionParam_1");
    lr_save_string("2222", "ExtensionParam_2");

    lr_xml_set_values("XML={XML_Input_Param}",
        "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
        "SelectAll=yes", "Query=//extension", LAST);

    NumOfValues= lr_xml_get_values("XML={NewXmlParam}",
        "ValueParam=OutputParam", "Query=//extension",
        "SelectAll=yes", LAST);

    for (i = 0; i < NumOfValues; i++) { /* Print the multiple values of MultiParam */

        sprintf(buf, "Retrieved value %d : {OutputParam_%d}", i+1, i+1);
        lr_output_message(lr_eval_string(buf));
    }

    return 0;
}

```

Output:

```

Action.c(40): Retrieved value 1: 1111
Action.c(40): Retrieved value 2: 2222

```

Specifying XML Function Parameters

Most XML API functions require that you specify the **XML element** and a **query**. You can also indicate if you want to retrieve all results or a single one.

Defining the XML Element

For defining the XML element to query, you can specify a literal string of the XML element, or a parameter that contains the XML. The following example shows the XML input string defined as a literal string:

```
"XML=<employee>JohnSmith</employee>"
```

Alternatively, the **XML** string can be a parameter containing the XML data. For example:

```
"XML={EmployeeNameParam}"
```

Querying an XML Tree

Suppose you want to find a value within an XML tag, for example, an employee's extension. You formulate a query for the desired value. The query indicates the location of the element and which element you want to retrieve or set. The path that you specify limits the scope of the search to a specific tag. You can also search for all elements of a specific type under all nodes below the root.

For a specific path, use "Query=*/full_xml_path_nameelement_name*"

For the same element name under all nodes, use "Query=*//element_name*"

In the VuGen implementation of XML functions, the scope of a query is the entire XML tree. The tree information is sent to the Vuser API functions as the value of the *xml* argument.

Multiple Query Matching

When you perform a query on an XML element, by default VuGen returns only the first match. To retrieve multiple values from a query, you specify the "SelectAll=yes" attribute within your functions. VuGen adds a suffix of *_index* to indicate multiple parameters. For example, if you defined a parameter by the name *EmployeeName*, VuGen creates *EmployeeName_1*, *EmployeeName_2*, *EmployeeName_3*, and so on.

```
lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);
```

With functions that *write* to a parameter, the values written to the parameter can then be evaluated. For example, the following code retrieves and prints multiple matches of a query:

```
NumOfValues = lr_xml_get_values("Xml={XmlParam}", "Query=//name",
"SelectAll=yes", "ValueParam=EmployeeName", LAST);
```

For functions that *read* from parameters, the values of the parameters must be pre-defined. The parameter must also use the convention *ParamName_IndexNumber*, for example *Param_1*, *Param_2*, *Param_3*, and so on. This collection of parameters is also known as a parameter set.

In the following example, `lr_xml_set_values` reads values from the parameter set and then uses those values in the XPath query. The parameter set that represents the employee extensions, is called `ExtensionParam`. It has two members: `ExtensionParam_1` and `ExtensionParam_2`. The `lr_xml_set_values` function queries the XML input string and sets the value of the first match to 1111 and the second match to 2222.

```
lr_save_string("1111", "ExtensionParam_1");
lr_save_string("2222", "ExtensionParam_2");

lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);
```

XML Attributes

VuGen contains support for attributes. You can use a simple expression to manipulate attributes of XML elements and nodes, just as you can manipulate the elements themselves. You can modify the desired attribute or only attributes with specific values.

In the following example, `lr_xml_delete` deletes the first cubicle element with the name attribute.

```
lr_xml_delete("Xml={ParamXml}",
             "Query=//cubicle/@name",
             "ResultParam=Result",
             LAST
            );
```

In the next example, `lr_xml_delete` deletes the first cubicle element with a name attribute that is equal to Paul.

```
lr_xml_delete("Xml={ParamXml}",
             "Query=//cubicle/@name="Paul",
             "ResultParam=Result",
             LAST
            );
```

Structuring XML Scripts

Initially, you create a new script in your preferred protocol. You can record a session in that protocol, or you may program the entire script without recording. Structure the Actions section of the script as follows:

- XML input declaration
- The Actions section

The XML input section contains the XML tree that you want to use as an input variable. You define the XML tree as a char type variable. For example:

```
char *xml_input=
"<acme_org>"
  "<employee>"
    " <name>John Smith</name>"
    "<cubicle>227</cubicle>"
    "<extension>2145</extension>"
  "</employee>"
  "<employee>"
    "<name>Sue Jones</name>"
    "<cubicle>227</cubicle>"
    "<extension>2375</extension>"
  "</employee>"
"</acme_org>";
```

The Action section contains the evaluation of the variables and queries for the element values. In the following example, the XML input string is evaluated using **lr_save_string**. The input variable is queried for employee names and extensions.

```
Action() {

  /* Save the input as a parameter.*/
  lr_save_string(xml_input, "XML_Input_Param");

  /* Query 1 - Retrieve an employee name from the specified element.*/
  lr_xml_get_values("XML={XML_Input_Param}",
    "ValueParam=OutputParam",
    "Query=/acme_org/employee/name", LAST);

  /* Query 2 - Retrieve an extension under any path below the root.*/
  lr_xml_get_values("XML={XML_Input_Param}",
    "ValueParam=OutputParam",
    "Query=//extension", LAST);

  return 0;
}
```

Enhancing a Recorded Session with XML

You can prepare an XML script by recording a session and then manually adding the relevant XML and Vuser API functions.

The following example illustrates how a recorded session was enhanced with Vuser API functions. Note that the only function that was recorded was **web_submit_data**, which appears in bold.

The first section contains the XML input declaration of the variable SOAPTemplate, for a SOAP message:

```
#include "as_web.h"

// SOAP message
const char*pSoapTemplate=
    "<soap:Envelope xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
    "  <soap:Body>"
    "    <SendMail xmlns=\"urn:EmailPortTypeInft-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.aspx/GetWeather",
                    "Method=GET",
                    "EncType=",
                    "RecContentType=text/xml",
                    "Referer=http://glkev.net.innerhost.com
                    /glkev_ws/WeatherFetcher.aspx?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:EmailPortTypeInft-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:EmailPortTypeInft-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;
}

```

Tasks

How to Use Result Parameters

Some of the `lr_xml` functions return a result parameter, such as **ResultParam**. This parameter contains the resulting XML data after the function is executed. The result parameters will be available from the parameter list in the Select or Create Parameter dialog box.

For example, for `lr_xml_insert`, `ResultParam` contains the complete XML data resulting from the insertion of the new XML fragment

You can use the result parameters as input to other XML related functions such as Web Service calls. During replay, VuGen captures the value of the result parameter. In a later step, you can use that value as an input argument.

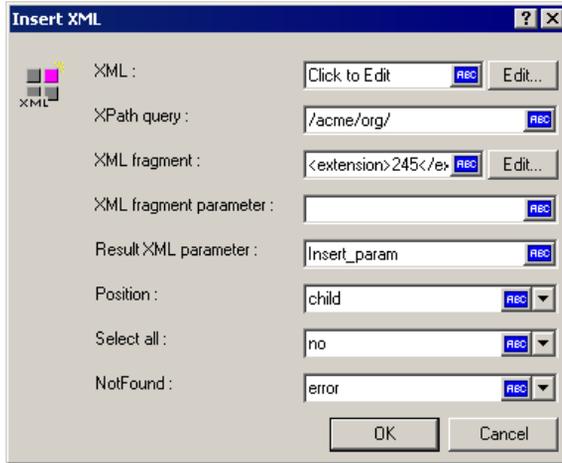
The functions that support result parameters are `lr_xml_insert`, `lr_xml_transform`, `lr_xml_replace`, `lr_xml_delete`, and `lr_xml_set_values`.

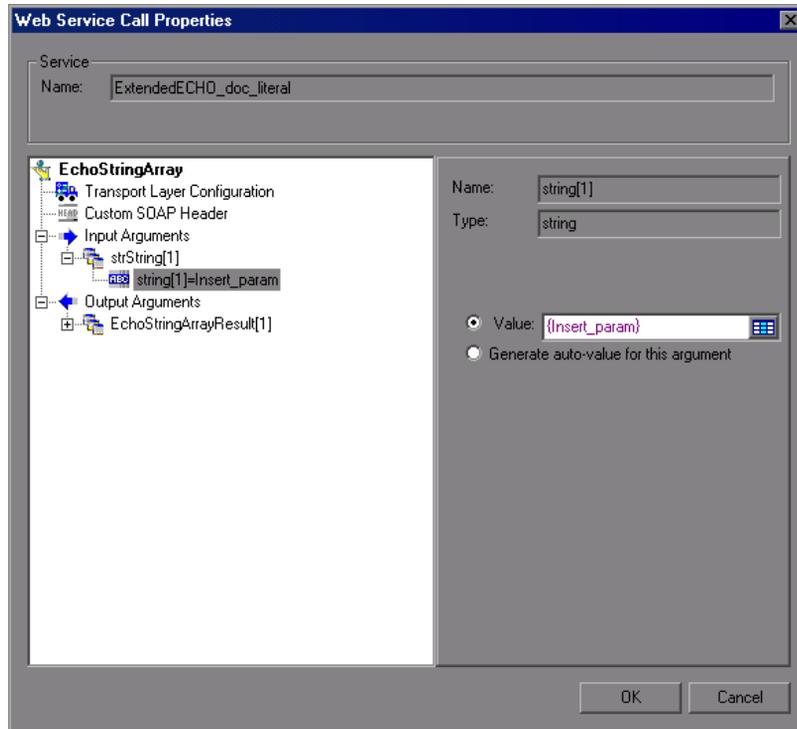
The following functions save values to a parameter other than the `resultParam`: `lr_xml_get_values` saves values to `ValueParam` and `lr_xml_extract` saves values to `XMLFragmentParam`. These values are also available for parameter substitution.

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 Arguments Node" on page 948.

Index

Symbols

.NET
 serialization 386

Numerics

164140 102
-25210 1405

A

ABC icon 340
abstract types 1070
accept server-side compression 565
Accept-Language request header 564, 1417
Acrobat Reader 50
Action
 method (Java) 837
 section 125
actions
 importing 130
Adobe Reader 21
Advanced GUI dialog box 437
advanced settings
 scenario 1276
agent for citrix presentation server 666
Agent for Microsoft Terminal Server
 tips 930
algorithm, for encryption 1278
allocating Vuser values
 data files 337
ALM 299
 connecting to 303
 managing scripts with 299
 managing Vuser scripts 300
amf
 code generation 396
 recording options 396
AMF terms 643
AMF Vuser scripts
 correlating 261
 overview 642
 recording scripts 642
 setting recording mode 646
analyzing run results. See run results
animated run
 defined 82
ANSI C support, in custom scripts 1400
append snapshot 935
Application Deployment Solution, Citrix
 Vuser type 653–686
Application Lifecycle Management 299
arrays
 duplicate in XML 334
aspects
 list 1116
 testing 1074
assemblies, adding in .NET 472
assertion, SAML 1262
asynchronous call, HTTP 1199
asynchronous messages 1199
attachments
 Web Service calls 1068
AUT Configuration 585
authentication
 connecting to Quality Center 206
 username (message) 1271
 username (transport) 1272
authentication during recording 121
authentication for WSDLs 1136
authentication retry think time 568
auto recovery 82
automatic synchronization 661
automatic transactions

Index

- transaction names 562
- Automatically 1086
- automation compliant 1399
- Axis/Java based Web services 1117

B

- Base 64 Process Complex dialog box 1114
- Base 64 Process dialog box 1113
- Base64 1073
- binary encoding 1277
- binary view of data (WinSock) 1358
- bitmap mismatch 682, 935
- block size, allocating Vuser values 337
- bookmarks 161
 - in Vuser script 150
- bookmarks, WinSocket data 1381
- books online 50
- Bootstrap policy 1273
- boundaries, defining for correlation 247
- Boundary testing
 - extreme values 1118
 - null values 1118
- BPT
 - parameter properties 357
- braces, using in parameterization 81
- Breakpoint Manager 165
- breakpoints 150, 159
- browser cache (Web, Wireless) 537
- bubbling 445
- bubbling, in Web events 445
- buffer data, editing 1364
- buffer size on network (Internet) 567
- buffer snapshot, WinSock data 1358

C

- C functions
 - additional keywords 152
 - for debugging 151
 - using in Vuser scripts 41
- C language support
 - conventions 1400
 - interpreter 42
- C Users 1399
- cache

- check for newer versions 537
- clear each iteration 538
 - loading and dumping 1042
- capture files 1077
- Certificate Authentication 1271
- Checkpoints 1228
- checkpoints
 - advanced 1146
 - expected values 1192
- Checkpoints tab 1248
- checkpoints, Web Services 1191
- checks (Web)
 - image checks 1052
 - overview 1043
- choice optional elements 1071
- Citrix
 - code generation settings 398
 - recording options 398, 401
- citrix
 - agent for citrix presentation server 666
 - recording options 399, 404
- Citrix agent 666
- citrix NFUSE 655
- citrix protocol 655
 - automatic synchronization 661
 - recording tips 656
 - synchronization 660
- Citrix Vuser scripts 653–686
 - display settings 86
 - function 676
- Citrix, *See Volume II - Protocols*
- Clear Service Call Log dialog box 1348
- clipboard, in RDP 921
- code example
 - EJB 770
- code generation
 - Citrix 398
- code generation options
 - EJB 423
- COM
 - data types 712
 - overview and interfaces 710
- COM Vuser scripts
 - class context 712
 - creating object instances 715

- error checking 714
- IDispatch interface 718
- instantiating objects 715
- interface pointers 713
- log files for debugging 721
- retrieving an interface 716
- script structure 712
- selecting COM objects to record 721
- type libraries 711
- understanding 712
- COM, *See Volume II - Protocols*
- com/dcom
 - recording options 407, 413
- command line arguments
 - reading in Java Vuser scripts 845
 - UNIX Vuser scripts 156
- command prompt 155, 156
- Commands tab (Customize dialog box) 72
- Comment button 185
- comments
 - inserting into Vuser scripts 184
- comparison method
 - HTML vs. text 85
- comparison settings 1178
- comparison tool, configuring 55
- compiling
 - Vuser scripts in UNIX 1455
- components
 - run results. *See* run results
- compression headers, requesting 565
- concurrency settings, Web Services 585
- configuraions files, customizing 1219
- configuration files, Web Services 1243
- configuring
 - application security and permissions 895
- configuring, Service Emulation host 1341
- Connect dialog box (RTE) 967
- connecting to Quality Center 206
- connection
 - database 1250
- connection attempts, modifying (RTE) 601
- connection pooling 884
- connection settings 1136
- Connection Settings dialog box 1173
- Connection String Generator 1251
- connections, closing open ones (.NET) 897
- console
 - overview 1331
- console, service emulation 1332
- content check
 - limit errors 567
- content, adding to script 1081
- context sensitive help 50
- Controller
 - scenario 177
- converting
 - Web functions to Java 1052
- coordinate shifting (RDP) 927
- copy and paste
 - RTE Vusers 969
- Correlated Query tab
 - COM 272
 - Database 256
- correlating
 - built-in detection 232
 - functions (C) 292
 - functions (Java) 293
 - Java statements 234
 - Microsoft .NET scripts 259
 - modifying existing parameters 245
 - SWECCount 271
- correlation 229
 - advanced 457
 - wdiff 287
 - web (HTTP/HTML) protocol 453
- correlation rules
 - testings 459
- count expression 1146, 1193
- cross-site scripting (XSS) 1117
- CtLib
 - logging server messages 525
 - options 418
 - result set errors 733
- CtLib, *See Also Volume II - Protocols*
- cusotm configuration files 1219
- custom headers, protection 1283
- Custom Vuser types
 - C Vusers 1399
 - Java Vusers 1403
 - JavaScript Vusers 1401
 - VB Vusers 1404

Index

- VBScript Vusers 1402
- custom web event recording 442
- Customize dialog box
 - Commands tab 72
 - Keyboard tab 74
 - Options tab 75
 - Toolbars tab 72
 - Tools tab 73
- customizing
 - Web Service scripts 1215

D

- data assignment methods, in
 - parameterization 322
- data buffers
 - Tuxedo Vuser scripts 1029
- data file parameters
 - importing data from database 343
 - selecting data source 351
- data files
 - used for parameterization 319
- data grids
 - viewing, .NET 874
- data navigation, WinSock 1361
- Data Navigator 1379, 1380, 1381
- data table parameters
 - importing data from database 343
 - selecting data source 353
- Data Wizard, SQL statement 344
- data.ws file 1374
- database
 - add connection 1236
 - Web Services, check values 1210
 - Web Services, retrieved data 1204
 - Web Services, validate data 1208
- Database Connection dialog box 1250
- database integration, Web Services 1203
- database protocols
 - recording options 414
 - recording options, advanced 416
- Database scripts
 - correlating 255
- Database Vuser scripts
 - developing 726
 - handling errors 732
 - return codes 737
 - tips 738
 - viewing grids 728
- Database Vuser scripts, *See Also Volume II-Protocols*
- database, Service Emulation 1341
- datasets, performing actions on 1212
- Date/Time parameter type 320
- DbLib 725
- debugging
 - database applications 735
 - during replay 150
 - enabling debugging features 84
 - enabling for Web Vusers 84
 - Oracle applications 731
- decrypt text 171, 176
- deep correlation (Java) 235
- default response
 - emulated service 1320
- defining parameter properties
 - data files 351
 - tables 353
- delays, for emulated services 1319
- delimiter of columns
 - in data tables 355
- derived types 1070
 - multiple roots 1162
- device name (RTE) 601
- DHTML 444
- diagnostics
 - enabling in VuGen 550
- digital signatures 1260
- directory of script, opening 123
- disabling functions (SAPGUI) 983
- distinguished names 858
- DLLs, calling from a Vuser script 1426
- DN (LDAP) 858
- DNS caching
 - Web 566
- DNS, *See Volume II - Protocols*
- DOM memory allocation 574
- download filter 558
- duplex communication 876
- duplicate key violations
 - Oracle, MSSQL 740

E

- EBCDIC format 1355
- edit, WinSocket buffer data 1364
- editing XML 1141
- editor for XML 1183
- editor, security scenario 1302
- editor, setting font for 82
- editor, XML 1161, 1163
- EJB
 - instance 772
 - method 774
 - recording options 423
 - script example 770
- EJB Detector
 - about 771
 - command-line 762
- Emulated Service
 - workflow 1327
- emulated service
 - default response 1320
 - operation rules 1320
 - setting rules 1320
- Emulation 1319
- encoding
 - EUC 379
 - passwords 171
- encoding, for WS config. file 1277
- encrypt text 171, 176
- encrypted data for Web Services security 1260
- end method 836
- entropy mode 1279
- environment settings
 - Java 839
 - Tuxedo Vusers 1026
- Environment tab 82
- Ericom 943
- Error
 - 25210 1405
- error handling
 - COM Vuser scripts 714
 - global or local setting 522
 - modifying globally 732
 - modifying locally (severity level) 733
 - run-time setting 551
- error matches, limiting 567

- error message 188
- errors, generate snapshot on 551
- escape sequence 1376
- EUC encoding 379
- EUC-JP encoding
 - recording option 448
- event handler 444
- expected values in checkpoints 1146, 1192
- exporting
 - Screen Recorder movies 213
- exporting script
 - to Word file 54
- extended result set 418

F

- failed bitmap synchronization
 - RDP 682, 935
- Federation scenario 1311
- field demarcation characters 950
- FIELDTBLS environment setting 1027
- files, adding to script 185
- filter files, editing for .NET 888
- Filter Manager, working with 466
- filtering
 - downloaded resources 558
 - Java methods 806
- filters in .NET
 - defining 893
 - determining elements to include 891
 - guidelines for setting 890
 - selecting 463
 - setting 463
- Find XML dialog box 1184
- Firefox support 703
- Flash remoting 642
- FLDTBLDIR environment setting 1027
- Flex
 - externalizable objects 543
 - RTMP 780
- flex
 - code generation 424
 - recording options 424
- Flex scripts
 - correlating 261
- Flex Vuser scripts

Index

- about 778
 - XML tree query 780
- font in editor 82
- forced mapping 485
- format
 - of data in display buffer 1376
- Forms Listener 912
- FTP protocol 58
- FTP, *See Volume II - Protocols*
- full run-time trace 549
- functions
 - automatic word completion 44, 83
 - ctx (Citrix) 676
 - Java 838
 - lr (C functions) 40
 - syntax 45

G

- General options
 - Citrix Display tab 86
 - Correlation tab 85
 - Environment tab 82
- generate snapshot on error 551
- generic toolkit (deprecated) 1117
- Get Text tool, Citrix Vuser scripts 669
- Getting Started 21
- global directory 81
- Global Unique Identifier (GUID) 711
- go to command 150
- Go To Offset dialog box 1380
- graphs
 - enabling for Web 561
- grids
 - enabling in .NET 875
 - viewing 728, 874
- Group Name parameter type 320
- GUI Vuser scripts
 - tools for 53
- GUID 711

H

- handler 444
- handlers
 - Web Services, examples 1220

- header files 46
- headers
 - risky 449
- history object, support for 572
- hook files 821
- Host Configuration dialog box 1341
- Host Name toolbar 1341
- Host Selection dialog box 1340
- host suffix, filtering by 558
- host, server emulation 1324
- hosted by client, server 880
- hotkeys 62
- HP Software Support Web site 26
- HP Software Web site 27
- HTML
 - maximum parameter length 248
- HTML-based mode 427
- HTTP
 - buffer size (Web) 567
 - HTTP recording mode, WAP 608

I

- ica files 679
- identities, Web Service security 1275
- IDispatch interface 718
- If-Modified-Since header
 - Web 537
- IIOp 796
- image checks
 - Web Vuser scripts 1052
- IMAP, *See Volume II - Protocols*
- i-mode, *See Volume II - Protocols*
- Import Service dialog box 1174
- Import SOAP, SOAP, importing 1098
- importing
 - actions 130
 - data from a database 343
 - SOAP requests into script 1063
- importing services 1174
- Informix 725
- init method 836
- Instantiating COM objects 715
- intellisense 44
- internal data, parameterization 320
- Internet Messaging (IMAP) 864

Iteration Number parameter type 320
 iterations
 run-time settings 552
 simulating in Web Services 1284
 updating parameters for each 350,
 358, 360, 361

IUnknown interface 711

J

Jacada Vuser scripts
 recording 799

Jacada, *See Volume II - Protocols*

Java

 custom filters 806

Java plug-in 811

Java Vuser scripts 791
 debug options 499

Java Vusers

 correlating statements 234
 editing Java methods 836
 environment settings 839
 inserting rendezvous points 841
 programming 827
 recording tips 811

Java Vusers (custom)

 creating template 835
 using Java code 1403

Java Vusers scripts

 run-time settings 577

Java Vusers, *See Volume II-Protocols*

Java, *See Volume II-Protocols*

JavaScript Vusers 1401

JMS

 asynchronous 1199
 message structure 1198

JNDI properties

 advanced, context factory 757
 locating EJB home 770
 specifying 756

Jscript 382

K

keep-alive connections, Web 564
 Kerebros

 authentication 568
 keyboard mapping (RTE) 964
 keyboard shortcuts 62
 keywords, adding additional 152
 Knowledge Base 26

L

labels, service emulation 1349

language headers 1417

LDAP, *See Volume II - Protocols*

legacy Web service security 1255

level of script generation, RDP 491

libc functions, calling 1400

libraries, for scripting 607

load balancing, Oracle NCA 251

Load Generator Name parameter type 320

loading DLLs

 globally 1440

 locally 1438

 overview 1426

LoadRunner Analysis User's Guide 22

LoadRunner Controller User's Guide 22

LoadRunner Installation Guide 22

log

 setting detail level - UNIX 1452

log, service emulation calls 1348

longevity testing 1118

lrbin.bat utility 1413

M

Mailing Services protocols

 IMAP 864

 MAPI 865

 POP3 867

 SMTP 868

Mailing services, *See Volume II - Protocols*

Managing Services 1131

MAPI, *See Volume II - Protocols*

mapping keyboard 964

maximum length of HTML parameters 248

Media Player, *See Volume II - Protocols*

memory allocation for DOM 574

memory management 574

message signatures 1260

Index

- messages
 - sending to output 191
- META refresh 567
- Microsoft .NET
 - recording options 463, 473
- Microsoft .NET Vuser scripts
 - correlating 259
 - limitations 897
 - overview 873
 - run-time settings 583
 - troubleshooting 895
 - viewing data grids 874
- mkdbtmpl script (UNIX) 1449
- MMS
 - run-time settings 586
- movies of your run session
 - capturing and viewing 213
 - exporting 213
 - removing from the test results 213
- MS
 - Exchange protocol (MAPI) 865
- MTOM 1277
- MTS components 411
- multilingual support 1415–1424
 - parameter files 1419
- multiple protocol script
 - recording options 426
- multi-threading 551

- N**
- namedPipe 1311
- negative testing, Web Services 1194, 1213
- netTcp 1311
- New Emulated Service dialog box 1343
- New Label dialog box 1349
- New Web Service Call 1100
- nodeset 1146, 1193
- non-printable characters 1377
- NTLM
 - authentication 121
 - security 568

- O**
- ODBC recording 725

- online browser 153
- Operations tab 1135
- optional elements
 - excluding 332, 1142
- optional parameters 1070
- optional windows 1001
- optional windows (SAPGUI) 983
- options
 - CtLib 418
 - Ird log 419
- Oracle
 - recording 2-tier database 725
- Oracle application debugging 731
- Oracle Configurator 912
- Oracle NCA
 - run-time settings 589
- Oracle NCA Vuser scripts
 - check connection mode 914
 - correlating 251
 - recording guidelines 902
 - secure applications 912
 - servlet testing 912
- Oracle Web Applications 11i Vuser scripts
 - advanced GUI-based options 437
- Oracle, *See Also Volume II-Protocols*
- OrbixWeb 798
- OTA, Over-The-Air 1389
- output message 188
- output parameters
 - XML 1477
- Output window 842
 - RunTime Data tab 103
- Overload testing 1118
- Overview 1318

- P**
- Pacing settings 552
- PAP, Push Access Protocol 1389
- parameter
 - create xml 341
 - restore original string 343
 - undo 343
- parameter type
 - Date/Time 320
 - Group Name 320

- Iteration Number 320
- Load Generator Name 320
- Random Number 320
- Unique Number 321
- Vuser ID 321
- parameter types
 - BPT 322
 - data files 319
 - internal data types 320
 - random number 357
 - tables 320
 - unique number 359
 - user-defined functions, format 321
 - xml 320
- parameterization
 - assigning values from files and tables 322
 - brace style 340
 - COM, .NET, VB 373
 - creating a new parameter 339
 - data files 319
 - Java 339
 - modifying existing parameters 245
 - naming a parameter 340
 - overview 317
 - random sequence with seed 323
 - setting properties for data files 351
 - setting properties for tables 353
 - simulating 364
 - tables 320
 - Tuxedo scripts 328
 - updating with unique values 323
 - user-defined functions 321
 - using internal data 320
 - UTF-8 encoded 1419
 - xml 320
- Parameterization Options 81
- parameterization, replacing long string 436
- parameterizing
 - security elements in WS scripts 1282
 - security scenario 1297
- parameters
 - creating in Script view 339
 - creating in Tree view 339
 - optional 1070
- Password Encoder dialog box 171, 177
- password, encoding 171
- PeopleSoft Enterprise Vuser scripts
 - advanced GUI-based options 437
- PeopleSoft, *See Volume II - Protocols*
- PeopleSoft-Tuxedo Vusers, running 1026
- Performance testing
 - longevity 1118
 - overload 1118
 - scalability 1118
 - stress 1118
 - volume 1118
- persistent connections, Web 564
- Plain SOAP scenario 1315
- policy files 1262
- pooling of connections 884
- POP3 (Post Office) protocol 867
- POP3, *See Volume II-Protocols*
- port mapping 480
 - advanced 486
 - overview 377
- Positive Testing 1116
- PPG, Push Proxy Gateway 1389
- pragma mode 589
- private key 1295
- private security scenario 1265
- programming
 - in Visual Studio 1407
 - using templates 1449
 - using Visual Basic templates 1412
 - Vuser actions 1447
- properties of parameters
 - defining for BPT 357
 - defining for data files 351
 - defining for tables 353
- Protection Level 1278
- protocol
 - advisor 109
- protocol advisor 109
 - overview 110
 - troubleshooting 118
 - workflow 111
- protocol detection 109
- protocols *See Volume II - Protocols*
- proxy server
 - for WSDLs 1136
- push support

Index

Wireless and WAP 1389
push technology 1041

Q

Quality Center
connecting to a project 206
Query Builder 1154

R

radius, for synchronization 493
raise tolerance 935
Random Number parameter type 320
random parameter assignment 323
RDP

- advanced code generation 488
- code generation 491
- recording options 494
- run-time settings 596

RDP Agent
tips 930
troubleshooting 930

RDP agent
recording options 490

RDP Vuser scripts
synchronizing replay 924

RDP, *See Volume II - Protocols*
read only WinSock buffers 1358
realm, Web Service security 1281
RealPlayer 60

recording
Web Services 1062

recording at the cursor 997
recording levels 382, 441
recording options

- See Also Volume II - Protocols*
- advanced correlation 457
- advanced database options 416
- amf code generation 396
- Citrix code generation node 398
- citrix configuration node 399
- Citrix login node 401
- citrix recorder node 404
- classpath 460
- com/dcom filter node 407

- com/dcom options node 413
- corba options 496
- correlation 497
- custom web event recording 442
- database node 414
- EJB 423
- EJB code generation 423
- flex code generation node 424
- general protocol node 426
- general script node 433
- HTTP advanced node 446
- HTTP correlation node 453
- Java VM 462
- language selection 433
- log options 499
- Microsoft .NET recording node 463, 473
- port mapping 377
- port mapping node 480
- port mapping, advanced 486
- RDP 494
- RDP advanced code generation node 488
- RDP agent 490
- RDP basic code generation node 491
- RDP login node 494
- recorder 501
- RTE configuration node 506
- RTE node 507
- SAPGUI auto logon node 509
- SAPGUI code generation node 510
- SAPGUI general node 511
- script generation preferences 380
- serialization 386, 503
- test correlation rule 459
- translation table overview 1355
- WinSock 518

recording tips
citrix 656

recording Vuser scripts
AMF 642
CORBA sessions 797

recursive elements, in WSDL 1072
reduce tolerance 935
references, adding for .NET 472
regenerating Vusers

- all protocols 133
- Reliable Messaging 1277
- rendezvous points
 - Java Vusers 841
- replay
 - prepare for 1225
- replay, preparing for 1189, 1255
- Report wizard 1344
- reports
 - service emulation 1344
- REST services 1152
- Result Details tab, Test Results window 213
- result parameters, saving XML 1477
- results
 - XML validation 1150
- Results Summary report
 - sending custom messages 207
- results. See run results
- return codes
 - database 737
- RMI
 - recording over IIOP 796
- roots, multiple 1162
- row count, obtaining 740
- RTE
 - recording options 506, 507
 - run-time settings 600
- RTE Vuser scripts
 - mapping PC keyboard 964
 - overview 941
 - reading text from screen 951
 - recording 966
 - synchronizing 953
- RTE, *See Volume II-Protocols*
- RTMP 780
- RTS
 - think time 554
- RTS_Log 547
- rules
 - emulated Web services 1320
 - service emulation 1320
- run results 203
 - customizing display 205
 - schema 205
- run_db_vuser shell script 158
- running Vuser scripts

- animated mode 82
 - step by step 150
 - using VuGen 147
- run-time settings 521
 - See Also Volume II - Protocols*
 - .NET 583
 - Additional Attributes 546
 - browser emulation 535
 - citrix configuration 539
 - citrix synchronization 541
 - configuring manually 1450
 - content checking 556
 - ContentCheck node (Web) 556
 - download filters 558
 - Flex externalizable 543
 - Flex RTMP 545
 - Java 577
 - Java classpath 577
 - Java VM 578
 - JMS 580
 - Miscellaneous 550
 - MMS 586
 - network speed 588
 - Oracle NCA 589
 - preferences 559
 - Preferences - Advanced 562
 - proxy 575
 - RDP 596
 - RDP Advanced 592
 - RDP Agent 594
 - RDP Configuration 596
 - RDP Synchronization 598
 - RTE 600
 - Run Logic node 553
 - SAPGUI 602
 - screen properties 574
 - VBA 606
 - WAP 607
 - WAP gateway 607
 - WAP radius 611
- run-time viewer
 - display options 84
 - enabling in VuGen 153

S

Index

- safearray log (COM) 414
- SAML
 - signing assertion 1262
- SAP (Click and Script) Vuser Scripts 971
- SAP, *See Volume II-Protocols*
- SAPGUI
 - auto logon options 509
 - recording options 509, 510, 511
 - run-time settings 602
- SAPGUI Vuser scripts
 - inserting steps 998
 - recording at the cursor 997
 - run-time settings 983
 - snapshots 998
- Scalability testing 1118
- scenario
 - create from VuGen 177
 - parameter simulation 366
- scenario setting, advanced 1276
- scenario types
 - core 1267
 - optimization 1269
 - security 1267
 - WCF 1268
- schema, for run results 205
- screen properties, run-time setting 574
- Screen Recorder tab, Test Results window 213
- script example
 - EJB 770
- script folder, opening 123
- script generation preferences 380
- script level, RDP 491
- script, adding content 1081
- Search for UDDI dialog box 1176
- searching for text on screen (RTE) 951
- sections of a Vuser script 125
- secure server, analyzing data 1080
- security
 - add SAML 1288
 - add Set Security 1286
 - customize for Web Services 1289
 - for importing WSDLs 1136
 - Web Service 1255
 - Web Services customizing 1292
 - Web Services, encryption 1257
 - Web Services, examples 1306
- security exceptions, .NET 895
- security scenario
 - editor 1302
 - how to 1310
 - parameterize 1297
 - prepare for run 1282
 - private 1265
 - types 1266
- security scenarios
 - protecting 1283
- Security testing
 - cross-site scripting (XSS) 1117
 - SQL injection vulnerability 1117
- securtiy scenario
 - custom binding 1274
- Select Certificate dialog box 1304
- sequential parameter assignment 323
- serialization 386
- serialization (Java correlation) 238
- service emulation
 - clear log 1348
 - console 1331
 - host 1324
 - rules 1320
- Service Emulation reports 1344
- service interoperability testing
 - .NET Framework 1117
 - Axis/Java based Web services 1117
 - generic toolkit (deprecated) 1117
- Service name, WSDL Definition tab 1172
- services
 - analyze traffic 1075
 - importing 1137, 1174
 - management 1170
 - managing 1155
 - managing overview 1133
 - new web service call 1100
 - select for recording 1096, 1126
- Set Security Properties dialog box 1299
- Shift Japan Industry Standard (SIJS) 379
- shifted coordinates 927
- shortcuts 62
- show function 44
- show function syntax 45
- Siebel
 - See Also Volume II-Protocols*

- Siebel correlation library 265
 - signatures, Web Service messages 1260
 - silverlight protocol 1019
 - overview 1020
 - SMTP protocol 868
 - SMTP, *See Volume II-Protocols*
 - Snapshot tab 1245
 - snapshots
 - generate on error 551
 - multiple RDP 936
 - SAPGUI Vusers 998
 - save replay snapshot locally 562
 - SOA Test Generator 1086
 - SOA tools 1139
 - SOAP requests, importing 1063
 - Solaris
 - ASCII translations 519
 - Speed Simulation settings 588
 - SPN 1275
 - SQL injection vulnerability 1117
 - SQL Server Studio Express 1332
 - SQL statement 344
 - SSL Configuration 1129
 - SSL, testing Web Service 1313
 - Standard Compliance testing 1116
 - still images of your application, capturing
 - and viewing 212
 - Stress testing 1118
 - strings, replacing long with parameter 436
 - SubjectKeyIdentifier 1290
 - support information 50
 - support online 50
 - SWECCount, correlating 271
 - synchronization
 - citrix 660, 661
 - synchronization failure
 - Citrix 682
 - RDP 935
 - synchronization functions
 - generating for Citrix text 399
 - generating for RDP 924
 - synchronizing Vuser scripts
 - block-mode (IBM) terminals 954
 - character-mode (VT) terminals 958
 - overview (RTE) 953
 - rendezvous 186
 - waiting for terminal to be silent 962
 - waiting for text to appear (RTE) 960
 - waiting for the cursor to appear 958
 - syntax, show for function 45
 - system variables
 - RTE 960
 - Tuxedo 1026
- ## T
- tables
 - used for parameterization 320
 - task_encode_password 177
 - TE system variables 960
 - template
 - create 134
 - Java Vuser 835
 - open 134
 - programming in C, UNIX 1449
 - programming using Visual Basic 1412
 - templates 124
 - Terminal Services
 - Citrix Vusers 659
 - Terminal Setup dialog box 967
 - test aspects 1116
 - Test Results report
 - sending custom messages 207
 - Test Results window
 - Screen Recorder and Result Details
 - tabs 213
 - theme 207
 - testing aspects 1074
 - text
 - reading text from screen (RTE) 952
 - searching for text on screen (RTE) 951
 - text synchronization
 - Citrix 399
 - RDP 924
 - text view (WinSock) 1358
 - think time
 - inserting 174
 - recommended ratio for Siebel 1012
 - threshold, Database 416
 - threshold, WinSock 520
 - thread, main (Java programming) 831
 - thread-safe code 830

Index

- threshold for setTimeout, setInterval 572
- timeouts
 - Citrix connection 541
 - HTTP request 565
 - WAP connection 565
- timestamp (Database) 418
- tips
 - Database related 738
 - recording RDP 920
 - Web Service security scenario 1310
- tokens 1257
- tolerance level (RDP) 935
- Toolbars tab 72
- traffic
 - SSL information 1129
- traffic files 1075, 1077
- traffic information, providing 1127
- transactions
 - breakdown limitation for Oracle DB 126
 - nested 179
- translation table 1355
- Translation table settings 520, 1355
- translation, ASCII on UNIX 519
- transport
 - HTTP/S 1194, 1231
 - JMS 1195, 1197, 1229
- transport, customizing HTTP 1280
- Tree view
 - Web Services 1245
- troubleshooting
 - 2-tier database 738
 - Oracle applications 731
 - protocol advisor 118
- Troubleshooting and Knowledge Base 26
- troubleshooting, .NET 895
- TUXDIR environment setting 1027
- Tuxedo Vuser scripts
 - data buffers 1029
 - environment settings 1026
 - log file 1030
 - system variables 1027
 - viewing data files 1029
- typing style (RTE Vusers) 947

U

- UDDI
 - information 1136
 - search for 1176
- UDDI Search dialog box 1176
- undo
 - parameter 343
- undo buffer, emptying (WinSock) 1365
- unique column name 744
- Unique Number parameter type 321
- unique value parameter assignment 323
- UNIX
 - command line 156
- update methods
 - parameter assignment 324
 - parameter usage 350
- UPN 1276
- URL-based mode 427
- user handler
 - create 1238
- user handlers 1216
- user-defined function parameters
 - format 321
- Username (Transport Protection) Authentication 1272
- Username Authentication (Message Protection) 1271
- UTF-8 conversion
 - in run-time settings 566
 - recording option 448

V

- validate XML 1158
- Validate XML dialog box 1179
- validation
 - checkpoints 1248
- value sets 1142
 - creating 331
 - optional elements 332, 1142
- VB Vusers 1404
- VBScript Vusers 1402
- verification checks
 - sapgui 1001
 - Web 560
 - Web (Click and Script) 693
- verify_generator 157

- Visigenic 798
 - Visual Basic
 - scripting option 382
 - Vuser scripts 1407
 - Visual C, using Visual Studio 1407
 - Visual Studio 1407
 - viewing scripts 886
 - Volume testing 1118
 - VuGen
 - environment options 82
 - recording Vuser scripts 119
 - Vuser functions
 - automatic word completion 44, 83
 - ctx (Citrix) 676
 - Flex 787
 - general (C) 40
 - Java 838
 - lr (C functions) 40
 - lrc (COM) 715
 - syntax 45
 - See Also* Online Function Reference
 - Vuser ID parameter type 321
 - Vuser information, obtaining 190
 - Vuser information, obtaining (Java) 842
 - vuser script template 134
 - Vuser script templates 124
 - Vuser scripts
 - ALM integration 299
 - C support 1400
 - comments, inserting 184
 - creating on UNIX ??–1456
 - importing from zip 127
 - Java language recording 791
 - opening 127
 - parameterizing 317
 - programming ??–1456
 - regenerating 133
 - running 147
 - running from command prompt 155, 156
 - sections 125
 - transactions 172
 - UNIX, compiling on 1455
 - UNIX, running on 156
 - working from zip 127
 - Vuser types
 - See Also* Volume II-Protocols
 - Java 791
 - Java (programming) 827
 - Real Player 60
 - vuser_init, vuser_end sections 125
- ## W
- waiting, for terminal to stabilize(RTE) 962
 - WAP
 - run-time settings 607
 - wap toolkits 1388
 - WAP, *See* Volume II-Protocols
 - Wdiff 244
 - wdiff 287
 - Web (Click and Script)
 - general options 571
 - history options 572
 - memory management 574
 - Navigator properties 573
 - timers 572
 - Web (Click and Script) Vuser scripts
 - about 1037
 - advanced GUI-based options 437
 - checking Web page content 556
 - debugging features, enabling 84
 - debugging tools 153
 - run-time viewer 153
 - troubleshooting tips 692
 - verifying text and images 1043
 - web (HTTP/HTML)
 - correlation 453
 - recording options 453
 - web event recording 441
 - Web performance graphs
 - generating for Web Vusers 561
 - Web Service call, new 1100
 - Web Service calls
 - adding new 1062
 - Web Service security
 - adding 1255
 - customizing 1292
 - Web Services
 - analyzing secure data 1080
 - customizing behavior 1215
 - database actions 1212

Index

- database values 1210
- HTTP/S transport 1231
- JMS transport 1229
- negative testing 1194, 1213
- run-time settings 580
- testing method, defining 1234
- testing methods 1214
- tools 1139
- transport HTTP/S 1194
- transport JMS 1195, 1197
- user handler examples 1220
- user handlers 1216
- validate data 1208
 - WSDL dependencies 1157
- Web Services security, custom 1274
- Web Services security, Federation 1273
- Web Services Vuser Scripts
 - add content 1059
- Web Services Vuser scripts
 - message signatures 1260
 - recording 1062
- Web Services, *See Volume II-Protocols*
- Web to Java converter 1052
- Web Vuser scripts
 - See Also Volume II-Protocols*
 - checks 1043
 - debugging features, enabling 84
 - debugging tools 153
 - image checks 1052
 - run-time viewer 153
 - setting Visual Log options 153
 - understanding 1036
 - verifying text and images 1043
- web_set_user function generation 121
- wildcards, Citrix window names 658
- Windows Authentication 1271
- Windows Sockets Vuser scripts
 - excluding sockets 519
 - viewing data files 1356
- Windows store 1304
- WinInet engine (Internet protocols) 562
- WinSock data, navigating 1361
- WinSock, *See Volume II-Protocols*
- WinSocket buffers, viewing 1369
- Wireless, *See Volume II-Protocols*
- wizard, recording WS 1096, 1126

- word completion 44, 83
- WS-Addressing 1200
- WS-Addressing version 1315
- WSDL
 - dependencies 1157
 - list of operations 1135
- WSDL file
 - managing 1172
- WSFederationHttpBinding 1279
- WsHttpBinding 1270
- WS-Reliable Messaging 1277
- WS-SecureConversation 1284
- WS-Security 1278
- WS-Security, customizing 1292
- WSxxx Tuxedo variables 1027

X

- X.509 certificate 1275
- X.509 certificates 1280
- XML
 - attributes, working with 1466
 - XML / WSDL Comparison 1178
- XML API
 - programming with 1459
- XML editing 1183
- XML editor 1141, 1161, 1163
- xml parameter
 - create from web service call 341
- XML parameters
 - creating 329
- XML queries 1154
- XML query 1184
- XML validation 1158, 1179
 - results 1150
- XP window style, Citrix 657
- XPATH
 - checkpoints 1193
- XPATH syntax 1146, 1193
- XSD schema, validation 1144
- X-SYSTEM message (RTE) 954

Z

- zip file
 - using 129

working from 127
zlib headers 565

