

HP Virtual User Generator

for the Windows operating system

Software Version: 9.50

User Guide Volume II - Protocols

Manufacturing Part Number: T7182-90020

Document Release Date: January 2009

Software Release Date: January 2009



Legal Notices

Warranty

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

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Third-Party Web Sites

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

Copyright Notices

© Copyright 2000 - 2009 Mercury Interactive (Israel) Ltd.

Trademark Notices

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

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

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

UNIX® is a registered trademark of The Open Group.

Documentation Updates

This guide's title page contains the following identifying information:

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

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

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

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

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

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

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

Support

You can visit the HP Software Support web site at:

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

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

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

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

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

To find more information about access levels, go to:

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

To register for an HP Passport ID, go to:

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

Table of Contents

Chapter 1: Understanding Protocols	17
Using This Guide	17
Vuser Types.....	18
Chapter 2: Web Services Protocol	21
About Web Services Scripts	21
Getting Started with Web Services Vuser Scripts	22
Creating an Empty Web Services Script	24
Viewing and Editing Scripts	25
Parameterizing Scripts	28
XML Editing	29
Chapter 3: Web Services - Managing	39
About Managing Web Services Vuser Scripts	40
Viewing and Setting Service Properties	41
Importing Services	45
Specifying a Service on a UDDI Server	48
Choosing a Service from Quality Center	49
Specifying WSDL Connection Settings	50
Deleting Services.....	52
Comparing WSDL Files	52
Viewing WSDL Files	57
Chapter 4: Web Services - Adding Script Content	59
About Adding Content to Web Services Scripts.....	60
Recording a Web Services Script.....	60
Viewing the Workflow	65
Adding New Web Service Calls	67
Importing SOAP Requests	69
Using Your Script.....	72
Managing Data in Quality Center.....	73
Working with Service Test Management	73

Chapter 5: Web Services - Server Traffic Scripts	77
About Creating Server Traffic Scripts	77
Getting Started with Server Traffic Scripts	79
Generating a Capture File	80
Creating a Basic Script from Server Traffic.....	82
Specifying Traffic Information	83
Choosing an Incoming or Outgoing Filter	84
Providing an SSL Certificate.....	86
Chapter 6: Web Services Call Properties	89
About the Web Service Call View	89
Viewing Web Services SOAP Snapshots	90
Understanding Web Service Call Properties	93
Derived Types	103
Working with Optional Parameters	104
Base 64 Encoding.....	108
Attachments	113
Working with the XML	117
Chapter 7: Web Services - Preparing for Replay	121
About Preparing Web Services Scripts for Replay	121
Checkpoints.....	122
Web Services JMS Run-Time Settings	123
Using Web Service Output Parameters	126
Handling Special Cases.....	130
Chapter 8: Web Services - Database Integration	131
About Database Integration	131
Connecting to a Database	132
Using Data Retrieved from SQL Queries	135
Validating Database Values after a Web Service Call.....	138
Checking Returned Values Through a Database.....	140
Performing Actions on Datasets.....	142
Chapter 9: Web Services - Security	143
About Adding Security for Web Service Testing	143
Adding Security to a Web Service Call - a General Workflow	145
Security Tokens and Encryption	147
Setting SAML Options	152
Examples Using web_service_set_security	155
Customizing Your Security.....	159

Chapter 10: Web Services - Advanced Security and WS Specifications	163
About Advanced Security and WS Specifications	164
Choosing a Security Model	165
Setting the Security Scenario.....	166
Scenario Types	170
Specifying Scenario Information.....	172
Selecting a Certificate	177
Advanced Settings	179
Customizing Your Security Model	184
Simulating Users with Iterations.....	188
Tips and Guidelines.....	190
Chapter 11: Web Services - Transport Layers and Customizations .	195
About Testing Web Service Transport Layers.....	195
Configuring the Transport Layer	196
Sending Messages over HTTP/HTTPS.....	197
Understanding JMS	198
Sending Asynchronous Messages	202
Chapter 12: Web Services - User Handlers and Customization	213
About Customizing Web Service Script Behavior	213
Setting up User Handlers.....	214
User Handler Examples	219
Using Custom Configuration Files.....	223
Chapter 13: Web Services - Negative Testing	225
About Applying Testing Methodologies	225
Understanding the Testing Settings	226
Defining a Testing Method	227
Evaluating the SOAP Fault Value	229
Chapter 14: Java Protocols - Recording	231
About Recording Java Language Vuser Scripts.....	232
Getting Started with Java Vuser Scripts.....	233
Recording Java Events	235
Recording CORBA.....	238
Recording RMI over IIOP.....	239
Recording RMI.....	240
Recording a Jacada Vuser	240
Recording on Windows XP and Windows 2000 Servers	241

Chapter 15: Java - Managing Vuser Scripts	243
Understanding Java Vuser Scripts	243
Working with CORBA	245
Working with RMI.....	247
Working with Jacada	248
Running a Script as Part of a Package	249
Viewing the Java Methods	250
Manually Inserting Java Methods	252
Configuring Script Generation Settings.....	254
Java Custom Filters.....	258
Chapter 16: Java - Correlating	267
About Correlating Java Scripts	268
Standard Correlation	269
Advanced Correlation	269
String Correlation.....	271
Using the Serialization Mechanism	272
Chapter 17: Enterprise Java Beans (EJB) Protocol	279
About EJB Testing.....	279
Working with the EJB Detector.....	280
Creating an EJB Testing Vuser.....	285
Understanding EJB Vuser Scripts.....	289
Running EJB Vuser Scripts.....	295
Chapter 18: Citrix Protocol	299
About Creating Citrix Vuser Scripts	300
Getting Started with Citrix Vuser Scripts.....	301
Setting Up the Client and Server.....	302
Recording Tips.....	304
Setting the Citrix Display Settings	306
Viewing and Modifying Citrix Vuser Scripts	307
Synchronizing Replay.....	308
Understanding ICA Files	316
Using Citrix Functions	317
Tips for Replaying and Troubleshooting Citrix Vuser Scripts	318

Chapter 19: Citrix - Agent for Citrix Presentation Server	325
About the Agent for Citrix Presentation Server	325
Using the Agent for Citrix Presentation Server Features	326
Data Execution Prevention (DEP) and Citrix Performance	330
Installing the Citrix Presentation Server Agent	332
Effects and Memory Requirements of the Citrix Agent.....	333
Sample Script	333
Chapter 20: Remote Desktop Protocol (RDP)	335
About Microsoft Remote Desktop Protocol (RDP) Vuser Scripts	336
Recording Tips	336
Recording an RDP Vuser Script	337
Running RDP Vuser Scripts	339
Working with Clipboard Data.....	339
Synchronizing Replay.....	342
Chapter 21: RDP - Agent for Microsoft Terminal Server	351
About the Agent for Microsoft Terminal Server.....	351
Using the Agent for Microsoft Terminal Server Features	352
Installing the Microsoft Terminal Server Agent.....	355
Effects and Memory Requirements	356
Chapter 22: Database Protocols	357
About Developing Database Vuser Scripts	358
Introducing Database Vusers.....	359
Understanding Database Vuser Technology	360
Getting Started with Database Vuser Scripts.....	361
Using LRD Functions.....	362
Understanding Database Vuser Scripts	363
Working with Grids.....	366
Troubleshooting Database Protocols.....	368
Evaluating Error Codes.....	369
Handling Errors	370
Chapter 23: Database - Script Correlation.....	373
About Correlating Database Vuser Scripts	373
Scanning a Script for Correlations	374
Correlating a Known Value.....	376
Database Correlation Functions.....	378
Chapter 24: DNS Protocol.....	379
About Developing DNS Vuser Scripts	379
Working with DNS Functions	380

Chapter 25: Windows Sockets (WinSock) Protocol.....	381
About Recording Windows Sockets Vuser Scripts.....	382
Getting Started with Windows Sockets Vuser Scripts.....	383
Setting the WinSock Recording Options	384
Using LRS Functions.....	387
Working with Windows Socket Data	388
Viewing Data in the Snapshot Window	389
Navigating Through the Data	391
Modifying Buffer Data.....	394
Modifying Buffer Names	401
Viewing Windows Socket Data in Script View.....	402
Understanding the Data File Format.....	403
Viewing Buffer Data in Hexadecimal format	405
Setting the Display Format.....	408
Debugging Tips.....	411
Manually Correlating WinSock Scripts	412
Chapter 26: Programming a Script in the VuGen Editor	417
About Creating Custom Vuser Scripts.....	418
C Vusers	419
Using the Workflow Wizard for C Vuser Scripts.....	420
Java Vusers.....	422
VB Vusers	423
VBScript Vusers.....	425
JavaScript Vusers	426
Chapter 27: Programming Java Scripts.....	427
About Programming Java Scripts	428
Creating a Java Vuser	429
Editing a Java Vuser Script	429
Java Vuser API Functions	431
Working with Java Vuser Functions	433
Setting your Java Environment.....	439
Running Java Vuser Scripts	439
Compiling and Running a Script as Part of a Package.....	440
Programming Tips	441
Chapter 28: COM Protocol.....	443
About Recording COM Vuser Scripts	444
COM Overview	444
Getting Started with COM Vusers.....	446
Selecting COM Objects to Record	447
Setting COM Recording Options	450

Chapter 29: COM - Understanding and Correlating	459
About COM Vuser Scripts.....	459
Understanding VuGen COM Script Structure.....	460
Examining Sample VuGen COM Scripts.....	462
Scanning a Script for Correlations	468
Correlating a Known Value.....	470
Chapter 30: AJAX (Click and Script) Protocol	473
About Developing AJAX (Click and Script) Vuser Scripts.....	473
Recording an AJAX (Click and Script) Session	474
Understanding AJAX (Click and Script) Scripts	474
Chapter 31: AMF Protocol	477
About Developing AMF Vuser Scripts	477
Understanding AMF Terms	479
Working with AMF Functions.....	480
Correlating AMF Scripts	481
Viewing AMF Data.....	485
Understanding AMF Scripts.....	485
Chapter 32: FTP Protocol	489
About Developing FTP Vuser Scripts.....	489
Working with FTP Functions	490
Chapter 33: Flex Protocol	491
About Developing Flex Vuser Scripts	491
Working with Flex Functions.....	493
Correlating Flex Scripts	494
Viewing Flex Data.....	498
Setting Flex Step Properties	501
Working with Flex RTMP	502
Chapter 34: LDAP Protocol	503
About Developing LDAP Vuser Scripts.....	503
Working with LDAP Functions	504
Defining Distinguished Name Entries.....	506
Specifying Connection Options.....	507

Chapter 35: Microsoft .NET Protocol	509
About Recording Microsoft .NET Vuser Scripts	510
Getting Started with Microsoft .NET Vusers	511
Viewing Scripts in VuGen and Visual Studio.....	513
Viewing Data Sets and Grids	515
Correlating Microsoft .NET Scripts	516
Configuring Application Security and Permissions	519
Recording WCF Duplex Communication.....	523
Chapter 36: Microsoft .NET Filters	533
About Microsoft .NET Filters	533
Guidelines for Setting Filters	535
Setting a Recording Filter	539
Working with the Filter Manager.....	541
Chapter 37: Web (HTTP/HTML, Click and Script) Protocols	551
About Developing Web Level Vuser Scripts.....	551
Introducing Web Vusers.....	552
Understanding Web Vuser Technology	553
Selecting a Web Vuser Type	553
Getting Started with Web Vuser Scripts.....	557
Recording a Web Session.....	559
Converting Web Vuser Scripts into Java.....	560
Support for Push Technology.....	561
Chapter 38: Web (Click and Script) Tips	563
Recording Issues	563
Recording Tips.....	565
Replay Problems	567
Replay Tips	569
Miscellaneous Problems	570
Miscellaneous Tips	572
Enhancing Your Web (Click and Script) Vuser Script.....	573
Chapter 39: Web (HTTP/HTML, Click and Script) Functions	579
About Web Vuser Functions	580
Adding and Editing Functions	581
General Web (Click and Script) API Notes.....	583
Using Cache Data.....	585

Chapter 40: Web (HTTP/HTML, Click and Script) Text and Image Verification	589
About Verification Under Load	589
Adding a Text Check	592
Understanding Text Check Functions	594
Adding an Image Check	600
Defining Additional Properties	603
Chapter 41: Modifying Web and Wireless Vuser Scripts	605
About Modifying Web and Wireless Vuser Scripts	606
Adding a Step to a Vuser Script	607
Deleting Steps from a Vuser Script	608
Modifying Action Steps	609
Modifying Control Steps	626
Modifying Service Steps.....	629
Modifying Web Checks (Web only).....	630
Chapter 42: Web (HTTP/HTML) Correlation Rules	631
About Correlating Statements.....	631
Understanding the Correlation Methods.....	633
Using VuGen's Correlation Rules.....	634
Setting Correlation Rules.....	639
Testing Rules.....	642
Chapter 43: Web (HTTP/HTML) -Correlation After Recording	643
About Correlating with Snapshots	644
Viewing the Correlation Results Tab.....	645
Setting Up VuGen for Correlations	648
Performing a Scan for Correlations	651
Performing Manual Correlation	655
Defining a Dynamic String's Boundaries	660
Chapter 44: Web (HTTP/HTML) - Handling XML Pages	663
About Testing XML Pages.....	663
Viewing XML as URL Steps	664
Inserting XML as a Custom Request	667
Viewing XML Custom Request Steps	668

Chapter 45: Oracle NCA Protocol	671
About Creating Oracle NCA Vuser Scripts	672
Getting Started with Oracle NCA Vusers	673
Recording Guidelines	674
Enabling the Recording of Objects by Name	676
Oracle Applications via the Personal Home Page	679
Using Oracle NCA Vuser Functions	681
Understanding Oracle NCA Vusers	681
Testing Oracle NCA Applications.....	683
Correlating Oracle NCA Statements for Load Balancing.....	686
Additional Recommended Correlations	687
Recording in Pragma Mode	689
Chapter 46: SAPGUI Protocol	693
About Developing SAPGUI Vuser Scripts.....	694
Checking your Environment for SAPGUI Vusers	695
Creating a SAPGUI Vuser Script	706
Recording a SAPGUI Vuser Script.....	707
Inserting Steps Interactively into a SAPGUI Script	710
Understanding a SAPGUI Vuser Script.....	712
Enhancing a SAPGUI Vuser Script	716
Chapter 47: SAPGUI - Replaying Scripts	721
About Replaying SAPGUI Vuser Scripts	721
Replaying SAPGUI Optional Windows	722
SAPGUI Functions	723
Tips for SAPGUI Vuser Scripts	724
Troubleshooting SAPGUI Vuser Scripts	728
Additional Resources	730
Chapter 48: SAP (Click and Script) Protocol.....	731
About Developing SAP (Click and Script) Vuser Scripts	731
Recording a SAP (Click and Script) Session.....	732
Understanding SAP (Click and Script) Scripts.....	732
Chapter 49: SAP-Web Protocol	735
About Developing SAP-Web Vuser Scripts.....	736
Creating a SAP-Web Vuser Script	736
Understanding a SAP-Web Vuser Script.....	738
Replaying a SAP-Web Vuser Script	740

Chapter 50: Siebel-Web Protocol	741
About Developing Siebel-Web Vuser Scripts.....	741
Recording a Siebel-Web Session	742
Correlating Siebel-Web Scripts	743
Correlating SWECOUNT, ROWID, and SWET Parameters.....	750
Troubleshooting Siebel-Web Vuser Scripts	752
Chapter 51: RTE Protocol	757
About Developing RTE Vuser Scripts	757
Introducing RTE Vusers.....	758
Understanding RTE Vuser Technology	759
Getting Started with RTE Vuser Scripts.....	759
Using TE Functions	761
Working with Ericom Terminal Emulation	761
Mapping Terminal Keys to PC Keyboard Keys.....	763
Chapter 52: RTE - Recording	765
About Recording RTE Vuser Scripts.....	766
Creating a New RTE Vuser Script	766
Recording the Terminal Setup and Connection Procedure.....	767
Recording Typical User Actions	771
Recording the Log Off Procedure	772
Typing Input into a Terminal Emulator	772
Generating Unique Device Names	775
Setting the Field Demarcation Characters	776
Chapter 53: RTE - Synchronization	779
About Synchronizing Vuser Scripts.....	779
Synchronizing Block-Mode (IBM) Terminals.....	781
Synchronizing Character-Mode (VT) Terminals	784
Chapter 54: RTE - Reading Text from Terminal Screen	791
About Reading Text from the Terminal Screen	791
Searching for Text on the Screen	792
Reading Text from the Screen	792
Chapter 55: Mailing Services Protocols	795
About Developing Vuser Scripts for Mailing Services.....	795
Getting Started with Mailing Services Vuser Scripts	796
Understanding IMAP Scripts	797
Understanding MAPI Scripts	798
Understanding POP3 Scripts	800
Understanding SMTP Scripts.....	801

Chapter 56: Tuxedo Protocols	803
About Tuxedo Vuser Scripts	804
Getting Started with Tuxedo Vuser Scripts	805
Understanding Tuxedo Vuser Scripts	806
Viewing Tuxedo Buffer Data	809
Defining Environment Settings for Tuxedo Vusers	810
Debugging Tuxedo Applications	811
Correlating Tuxedo Scripts	811
Chapter 57: Real and Media Player Protocols	819
About Recording Streaming Data Virtual User Scripts	820
Getting Started with Streaming Data Vuser Scripts	820
Using RealPlayer LREAL Functions	821
Using Media Player MMS Functions	822
Chapter 58: Wireless Protocols	823
Understanding the WAP Protocol	823
Getting Started with Wireless Vuser Scripts	825
Using Wireless Vuser Functions	827
Push Support	828
VuGen Push Support	830
MMS (Multimedia Messaging Service) Vuser Scripts	831
Running an MMS Scenario in the Controller	832
Index	833

1

Understanding Protocols

VuGen supports a variety of applications and protocols, allowing you to record and create a script that accurately emulates your actions.

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

Using This Guide

This user's guide, *Protocols*, is the second volume of the HP Virtual User Generator user's guide. The first volume, *Using VuGen*, describes how to work with VuGen and create tests. This volume describes the unique settings and guidelines for the individual protocols. For example, this volume includes the recording options and run-time settings that are protocol-specific, while *Using VuGen* lists the settings that are common to all or most of the protocols.

When you are deciding which Vuser type to record, you may find that your application uses several protocols, such as Web & FTP or Web & Web Services. VuGen supports recording for multi-protocol scripts. For more information, see *Volume I-Using VuGen*.

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

To develop GUI Vuser scripts for use with LoadRunner, refer to the *WinRunner User's Guide* or QuickTest.

Vuser Types

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

The Vuser types are divided into the following categories:

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

- **Streaming.** For MediaPlayer (MMS) and RealPlayer protocols.
- **Wireless.** For Multimedia Messaging Service (MM) and WAP protocols.

2

Web Services Protocol

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

This chapter includes:

- About Web Services Scripts on page 21
- Getting Started with Web Services Vuser Scripts on page 22
- Creating an Empty Web Services Script on page 24
- Viewing and Editing Scripts on page 25
- Parameterizing Scripts on page 28
- XML Editing on page 29

About Web Services Scripts

SOA systems are based on Web Services, self-contained applications that can run across the Internet on a variety of platforms. The services are built using Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP). They serve as building blocks enabling the rapid development and deployment of new applications.

Using VuGen, you create test scripts for testing your SOA environment. You can use a test generation wizard to automatically generate scripts, or create the scripts manually.

To manually create scripts, you begin by creating an empty script. Then you add content to the script either by recording a session, analyzing network traffic, or manually inserting calls to the Web service as described in Chapter 4, "Web Services - Adding Script Content."

For manual scripts, you use VuGen to create one of the following scripts.

- ▶ **Single Protocol Script.** A script that emulates SOAP traffic by sending SOAP requests to the Web service.
- ▶ **Multi Protocol Script.** A script that emulates several protocols in a single script. For example, if your environment contains a client that accesses a Web Services and Web pages, select both the Web Services and Web (Click and Script) protocols.

Getting Started with Web Services Vuser Scripts

This section provides an overview of the process of developing a Web Services / SOA Vuser script.

To develop a test script:

1 Create a new Web Services script.

Create a new script using the SOA Test Generator, or manually create a new single or multiple protocol script, or a Business Process Testing component.

2 Add content to the script.

Add content to the script (excluding the SOA Test Generator). For details, see Chapter 4, "Web Services - Adding Script Content."

3 Set properties, values, and checkpoints.

Enhance the script by customizing the step properties, inserting argument values, and setting checkpoints. For details, see Chapter 6, "Web Services Call Properties."

4 Parameterize your script.

Parameterization lets you replace constant values with a variable to substitute new values for each iteration. To parameterize a value, double-click on a step to open its properties and click the **ABC** icon adjacent to the value box. For complex type elements, use the XML parameter type as described in Chapter 14, "File, Table, and XML Parameter Types" in *Volume I-Using VuGen*.

5 Enhance your script with transactions.

Define a group of actions as a transaction to check the applications's performance. For example, if you want to check the time it took for a service to update an address, you mark those actions as a transaction. For more information, see Chapter 6, "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

6 Configure the Run-Time settings.

The Run-Time settings control the script's behavior during execution. These settings include Web Service-specific settings (client emulation) and General settings—run logic, pacing, logging, and think time.

For information about the Web Service-specific settings, see "Web Services JMS Run-Time Settings" on page 123, and "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

7 Verify that the script is functional.

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

For details about replaying the script, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

8 Save the script.

Save the script in the file system or in a Quality Center repository. If you save the scripts in Quality Center, you can associate them to a test set and perform functional and regression testing directly from Quality Center. For more information about Quality Center and its integration with scripts, see "Working with Service Test Management" on page 73.

After you prepare a script, you are ready to use it for your testing. For more information, see "Using Your Script" on page 72.

Use Quality Center to manage all of your tests while tracking defects and requirements. For more information, see www.hp.com or contact your sales representative.

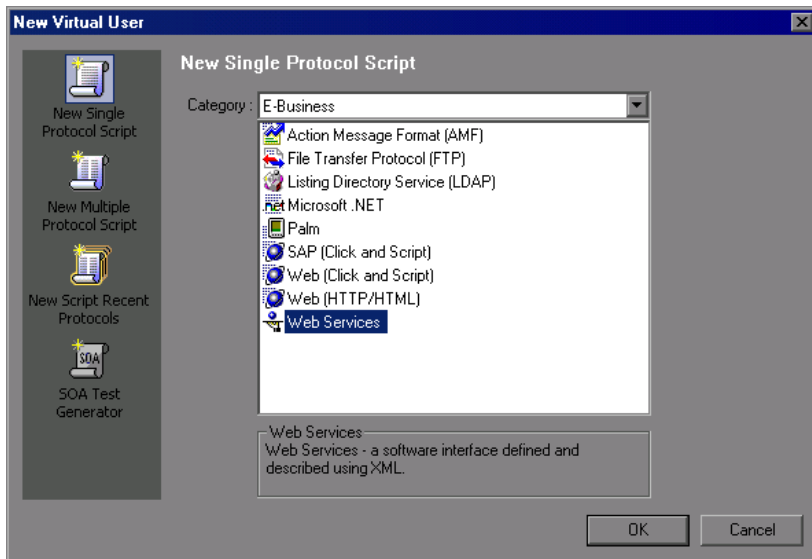
Creating an Empty Web Services Script

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

To create an empty script:



- 1 Create an empty script. Click the New Script button or select **File > New** to open the New Virtual User dialog box.
- 2 Click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the **E-Business** category. Click **OK**.



If you need to record several different protocols, such as Web Services and Web, click **New Multiple Protocol Script** in the left pane and specify the desired protocols. Click **OK**.

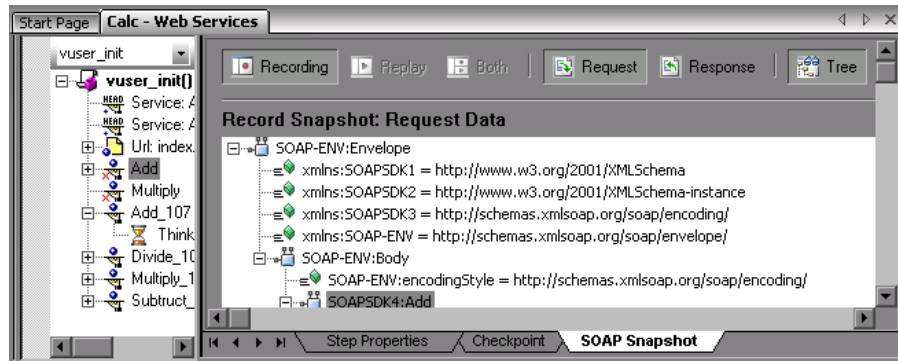
Viewing and Editing Scripts

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

You can view a script in either Tree View or Script View. Tree view displays the steps of the script in a graphical interface, while the Script view shows all steps, including the actual **web_service_call** functions that emulate your service. Script view is ideal for advanced users that require more flexibility within the script.

Tree View

The Tree view shows a graphical representation of each one of the script's steps.



When you select a step, VuGen displays information about the step in several tabs:

- ▶ **Step Properties.** The properties and argument values of the Web service call. This tab allows you to modify the properties of an existing step. See "Understanding Web Service Call Properties" on page 93.
- ▶ **Checkpoint.** A list of checkpoints defined for the step. See "Checkpoints" on page 122.
- ▶ **SOAP Snapshot.** A snapshot of the SOAP request and response for both record and replay. See "Viewing Web Services SOAP Snapshots" on page 90.

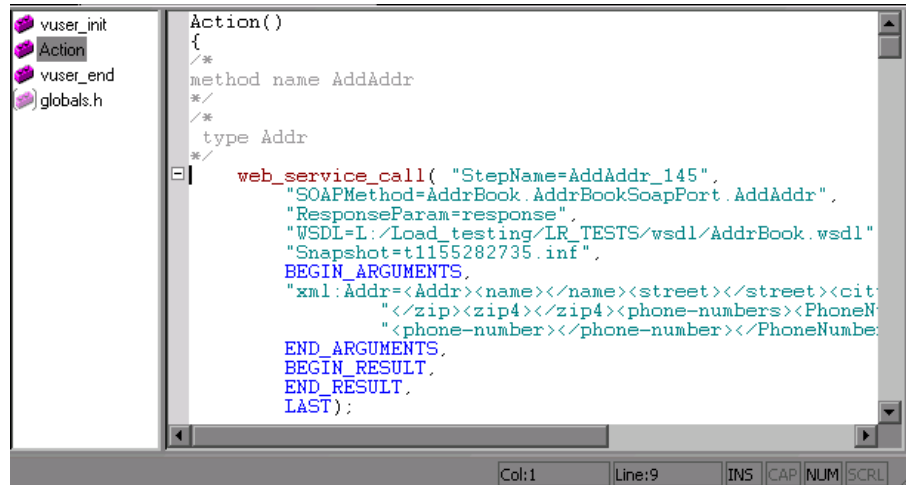
For more information about these tabs, see Chapter 6, "Web Services Call Properties."

To view a script in Tree view:

- 1** Click the **Tree** button or select **View > Tree View**.
- 2** In the upper left box, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new action select **Actions > Create New Action**.
- 3** In the left pane, select the step or sub-node that you want to view or modify.
- 4** Select the **Step Properties** tab in the right pane to view or modify the properties.
- 5** Select the **Snapshot** tab to view the step's SOAP header and body. To display a specific replay iteration, select **View > Snapshot > Select Iteration**.
- 6** To add additional Web Service steps, click the **Add Service Call** button. For more information, see "Adding New Web Service Calls" on page 67.
- 7** To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert > Add Step** and select the appropriate step. For more information, see Chapter 9, "Web Services - Security."
- 8** To replace argument values with parameters, go to the **Step Properties** tab. Select the node whose value you want to replace in the script, and click the **ABC** icon to the right of the **Value** box.
- 9** To set a checkpoint, click the **Checkpoint** tab. For more Information, see "Checkpoints" on page 122.

Script View

The Script view shows the actual functions that were generated in the script. You can expand or collapse each of the `web_service_call` functions to view only the functions that interest you.



To view a script in Script view:

- 1 Click the **Script** button or select **View > Script View**.
- 2 In the left pane, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new section select **Actions > Create New Action**.
- 3 To add additional Web Service steps at the location of the cursor, click the **Add Service Call** button. For more information, see Chapter 6, "Web Services Call Properties."
- 4 To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert > Add Step** and select the appropriate step. For more information, see Chapter 9, "Web Services - Security."
- 5 To replace argument values with parameters, select the value you want to replace in the script, and select **Replace with Parameter** from the right-click menu.

For more information about the functions, see the *Online Function Reference* (**Help > Function Reference**) or select a function and click F1.

Parameterizing Scripts

VuGen supports parameterization for all of the argument values. Parameterization lets you substitute the original values with external values. This is useful for testing your service with different values, or passing information from one step to another. For an overview on parameterization, see "Creating Parameters" in *Volume I-Using VuGen*.

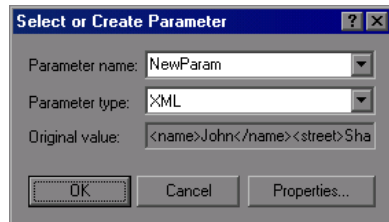
If your arguments are the simple, non-array type, you can replace them with a simple parameter. For example, if you want to test a service that does addition, you can substitute each of the input arguments with a parameter, and store the values in a file or a table.

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

Using parameters, you can pass the output value from one operation, as input for a later operation. For more information, see "Using Web Service Output Parameters" on page 126.

To replace a constant value with a parameter:

- 1 Switch to the **Step Properties** tab and select the parent or child element whose value you want to parameterize.
- 2 Under the **Input Arguments** node, select the argument you want to parameterize. In the right pane, click the **ABC** icon in the **Value** box. The Select or Create Parameter dialog box opens.



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

For more information, see "Creating Parameters" in *Volume I-Using VuGen*.

XML Editing

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

Schema	Set 1
[-] Addr	
[-] name	John Smith
[-] street	3 Acorn Lane
[-] city	Phoenix
[-] state	AZ
[-] zip	65354
[-] zip4	3333
[-] phonenumber	
[-] PhoneNumber [..]	
[-] PhoneNumber[1]	

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

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

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

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

Multiple Value Sets

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

Schema	Set 1	Set 2	Set 3
Addr			
name	John Doe	Tom Smith	Kim Jones
street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
city	Delray Beach	NIL	NIL
state	FL	AZ	MA
zip	33452	NIL	02134
zip4			
phonenumbers			
PhoneNumber [..]			
PhoneNumber[1]	NIL	NIL	NIL

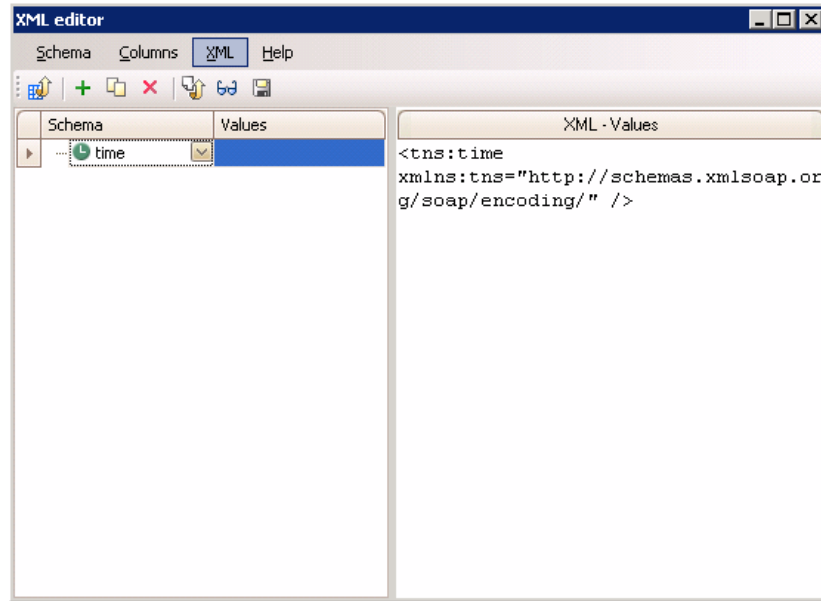
You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

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

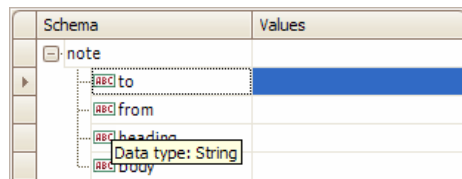
To resize the columns of the value sets, click the divider in the column's title and drag your mouse to the desired width.

To open the XML editor:

- 1 Choose **SOA Tools > XML Editor**. The editor opens.
- 2 To load a schema, select **Schema > Load**. To load an XML file, choose **XML > Load**.



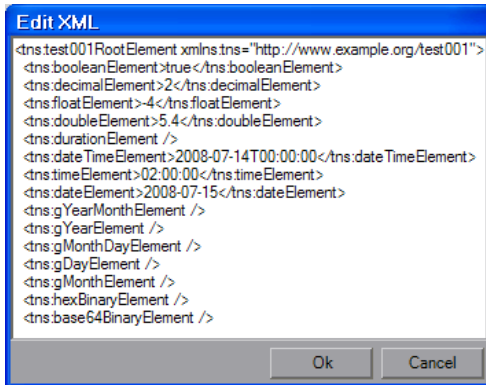
- 3 Fill in the **Values** column manually, or load values from an XML file. Click the **Load XML from file into the selected column** button. Place your mouse over the element's icon to discover its data type.



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



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



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



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



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

Arrays

You can manipulate array elements and duplicate their whole structure.

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

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

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

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

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

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

Duplicating Arrays

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

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

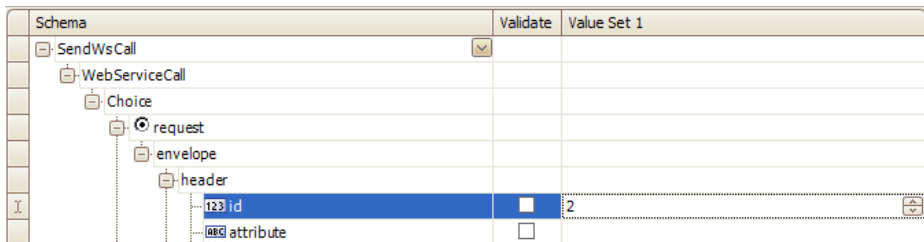
Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [..]			
PhoneNumber[1]	[◆]	[◆]	[◆]
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	[◆]	[]	[◆]
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneNumber[3]	[◆]	[]	[]
description	Mobile	Mobile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

Data Types

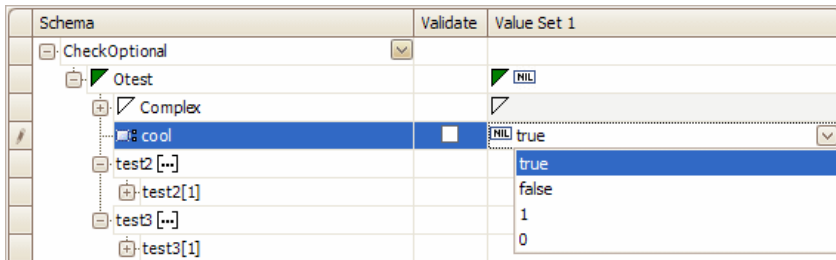
The XML editor lets you manipulate arrays of elements and sub-elements and assign them values.

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

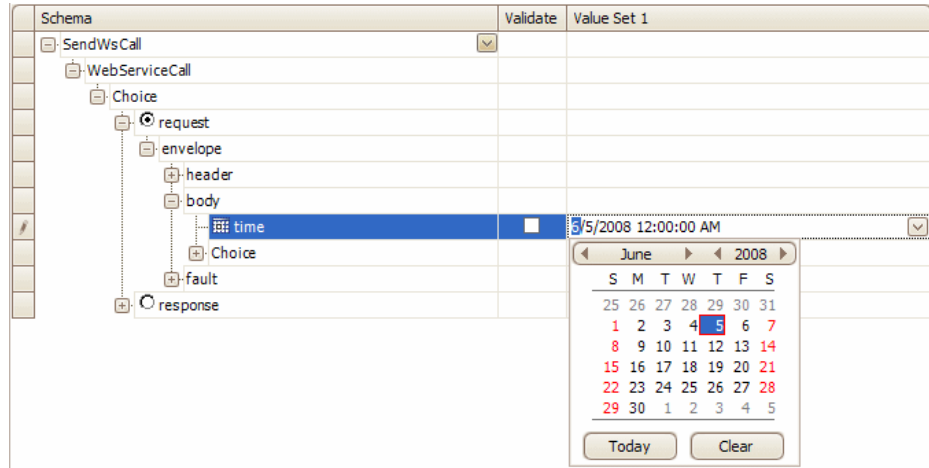
The grid recognizes element data types and provides an interface for setting the values. For example, for an **Int** type, the value cell contains a number scrolling control.



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



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



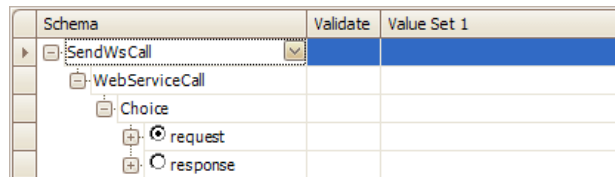
The schema indicates **Base64** elements with a **B64** button to the right of the value box. Click the button to open the Process Base 64 Data dialog box. For more information, see "Base 64 Encoding" on page 108.



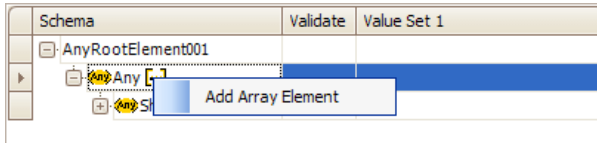
Any Type Elements

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

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

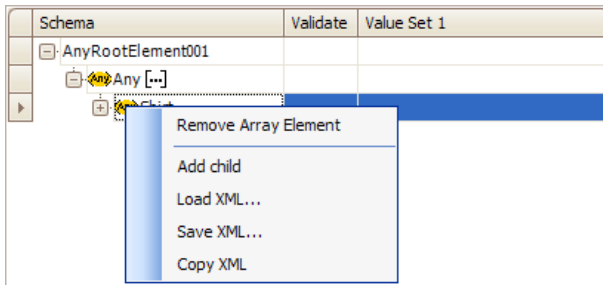


For an array of Any elements, you can manipulate the elements, their names, and their values. Use the right-click menu to add elements.



Manipulating Any Array Elements

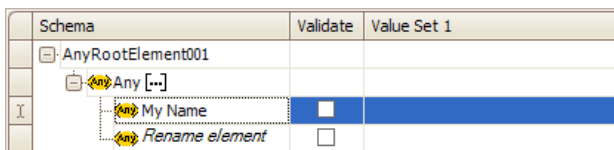
To manipulate Any array elements, open the right-click menu.



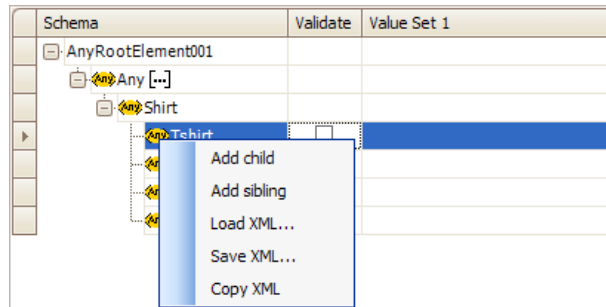
The right click menu provides some or all of the following options, depending on the location of the cursor:

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

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

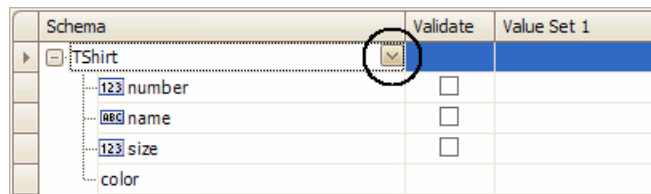


When manipulating a child element, you can use the right-click menu to add a sibling element.



Multiple Roots

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



XML Editor Integration

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

Parameterization uses the editor to display parameter elements and values as described in *Volume I-Using VuGen*.

3

Web Services - Managing

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

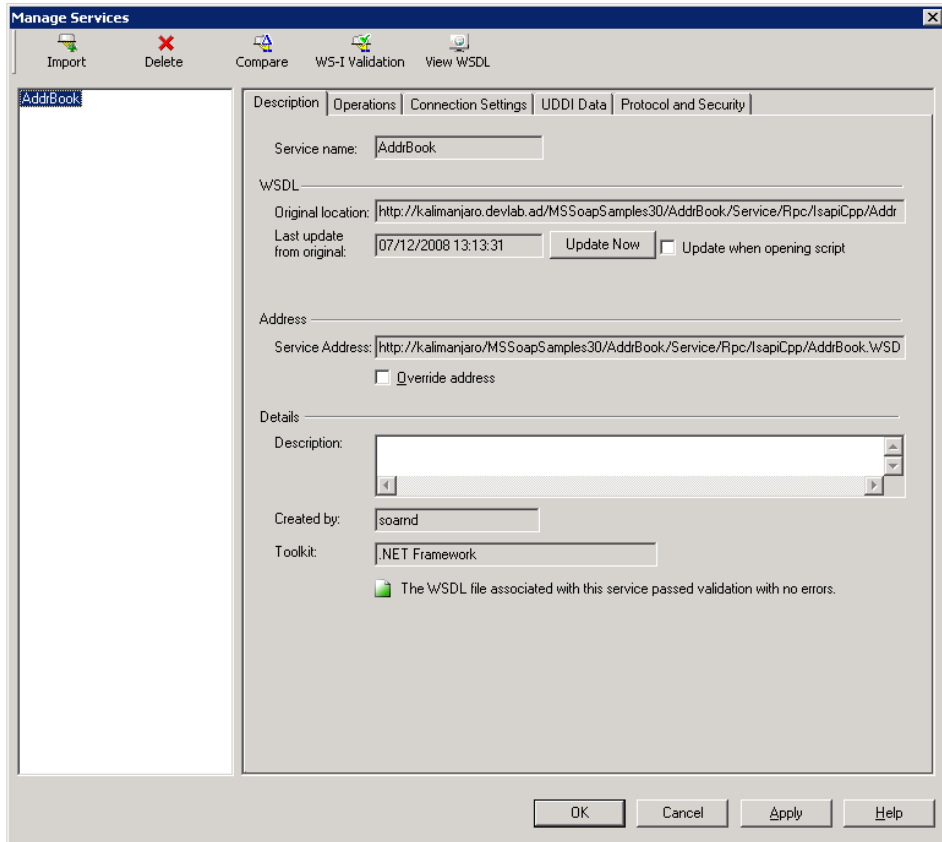
This chapter includes:

- ▶ About Managing Web Services Vuser Scripts on page 40
- ▶ Viewing and Setting Service Properties on page 41
- ▶ Importing Services on page 45
- ▶ Specifying a Service on a UDDI Server on page 48
- ▶ Choosing a Service from Quality Center on page 49
- ▶ Specifying WSDL Connection Settings on page 50
- ▶ Deleting Services on page 52
- ▶ Comparing WSDL Files on page 52
- ▶ Viewing WSDL Files on page 57

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

About Managing Web Services Vuser Scripts

The Service Management window lets you manage a list of service entries for the current script. You can view and set the properties of each service entry.



You add service entries to the list by importing WSDL files. When you add a WSDL to the list, VuGen creates a working copy that it saves with the script—it is not global. Therefore, for each script that you create, you must import the desired WSDL files.

To view the copy of the locally saved WSDL in Internet Explorer, click the **View WSDL** button.

Note: All validations and modifications to WSDL files are done on the working copy. If you want to replace the imported WSDL file with a newer version, use the **Update Now** option described in "Description" on page 41.

To open the Service Management window, select **SOA Tools > Manage Services** or click the Manage Services toolbar button.

The Service Management window provides an interface for:

- Viewing and Setting Service Properties
- Importing Services
- Deleting Services
- Comparing WSDL Files
- Viewing WSDL Files

Viewing and Setting Service Properties

The Service Management window lets you view and modify information about the imported WSDLs. It shows details about the selected service entry in the following tabs:

- Description
- Operations
- Connection Settings
- UDDI Data

Description

The Service Management window's **Description** tab displays information about the service: WSDL, Endpoint Address, and Details. You can add annotations or notes about your service, in the Details area.

WSDL

The WSDL area provides information about the location of the WSDL, and the date that it was last imported.

- **Original location.** The original source of the WSDL file (read-only).
- **Service name.** The name of the Web Service.
- **Last update from original.** (For services **not** imported from Quality Center) The last date that the local copy was updated with a WSDL file from the original source.
 - To manually update the working copy of the WSDL, click **Update Now**. VuGen backs up the existing WSDL and updates it from the location indicated above.
 - To instruct VuGen to update the WSDL every time you open the script, select **Update when opening script**.
 - date of the last of the service from QC
- **Last updated from QC.** (For services imported from Quality Center) The last date that the service was updated from Quality Center.

Address

- **Service address.** An endpoint address to which the request is sent.

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

Details

- **Description.** A description of the Web service, taken by default from the WSDL file. This text area is editable.
- **Created by.** The name of the user who originally imported the service (read-only).
- **Toolkit.** The toolkit associated with the script. You set this before importing the first WSDL file.

Operations

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

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

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

Connection Settings

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

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

- SSL

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

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

The screenshot shows the 'Connection Settings' tab of a Service Manager interface. It features two main sections: 'Authentication' and 'Proxy'. Each section has a checkbox to enable its respective settings and several text input fields. Below each section is a note stating that the values only apply to WSDL imports and that specific steps should be added to a replay for these values to take effect.

Authentication

Use Authentication Settings

Username:

Password:

The above values only apply to the importing of WSDLs. To use these values during replay, add a `web_set_user` step with the desired values.

Proxy

Use Proxy Settings

Server: Port:

Username:

Password:

The above values only apply to the importing of WSDLs. To use these values during replay, add a `web_set_proxy` step with the desired values.

For more information about setting the connection information while importing the WSDL, see "Connection Settings" on page 48.

UDDI Data

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

Description	Operations	Connection Settings	UDDI Data
UDDI server: <input type="text" value="http://ysayers2-il:8080/uddi/inquiry"/>			
UDDI version: <input type="text" value="2"/>			
Service key: <input type="text" value="527276d0-dc71-11db-bcc2-8bdaa861bcc2"/>			

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

For information about importing from a UDDI, see "Specifying a Service on a UDDI Server" on page 48.

Protocol and Security Settings

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

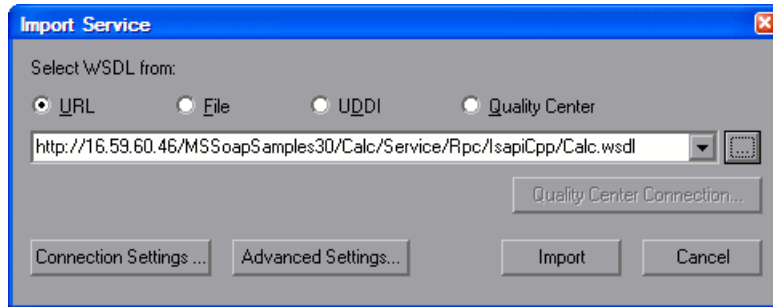
While this tab shows the information as Read-Only, click the **Edit Data** button to modify the security settings. For more information, see "Setting the Security Scenario" on page 166

Importing Services

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

When importing a file, you specify the following information:

- ▶ **Source.** the source of the WSDL: URL, File, UDDI, or Quality Center
- ▶ **Location.** the path or URL of the WSDL, entered manually or by browsing
- ▶ **Toolkit.** the toolkit to permanently associate with all services in the script (only available for the first service added to the script)
- ▶ **Connection Settings.** authentication or proxy server information (optional)



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

Source

When specifying a WSDL, you can indicate the source:

- ▶ **URL.** The complete URL of the service.
- ▶ **File.** The complete path and name of the WSDL file.
- ▶ **UDDI.** Universal Description, Discovery, and Integration—a universal repository for services. For more information, see "Specifying a Service on a UDDI Server" on page 48.
- ▶ **Quality Center.** A service stored in the Quality Center repository. For more information, see "Choosing a Service from Quality Center" on page 49.

VuGen supports URL or UDDI paths that are secure, requiring authentication or accessed through proxy servers. For more information, see "Specifying a Service on a UDDI Server" on page 48.

Location

In the Location box, you specify the path or URL of the WSDL.

For the URL or UDDI options, make sure to insert a complete URL—not a shortened version. Click the **Browse** button to the right of the text box to open the default browser.

For a file, click the **Browse** button to the right of the text box to locate the WSDL on the file system.

For Quality Center, click the **Quality Center Connection** button to specify a server URL and to initiate a connection. For more information, see "Choosing a Service from Quality Center" on page 49.

Toolkit Selection

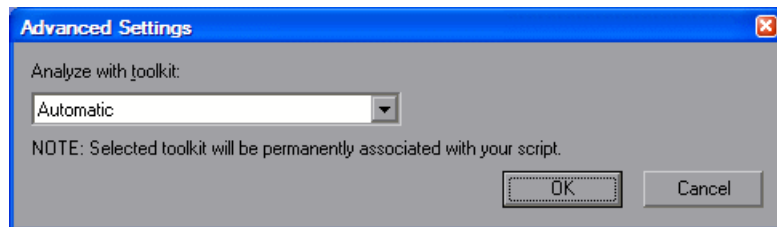
Choosing a toolkit instructs VuGen to send real client traffic using an actual toolkit—not an emulation. Once you select a toolkit, it becomes permanently associated with the script for all subsequent recordings, imports, and replays.

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

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

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

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



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

Connection Settings

When importing WSDL files from a URL or UDDI, the WSDL may require authentication if it resides in a secure location. In certain cases, the access to the WSDL may be through a proxy server. Using the Connection Settings button, you can specify this information. For more information, see "Specifying WSDL Connection Settings" on page 50.

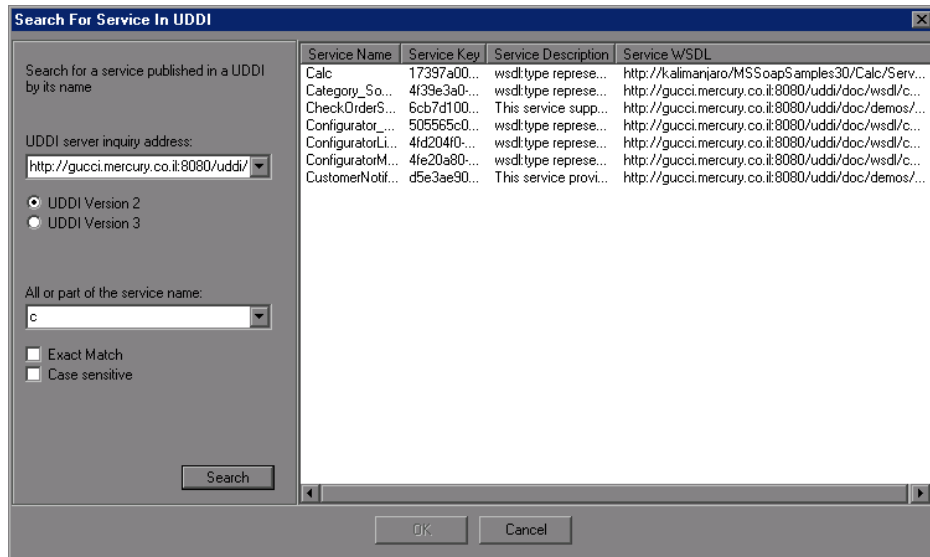
Specifying a Service on a UDDI Server

Service brokers register and categorize published Web Services and provide search capabilities. The UDDI business registry is an example of a service broker for WSDL-described Web Services.

Your Web Service client can use broker services such as the UDDI, to search for a required WSDL-based service. Once located, you bind to the server and call the service provider.



Click the **Browse** button to open the Search for Service in UDDI dialog box.



To search for a service on a UDDI:

- 1 In the **UDDI server inquiry address** box, enter the URL of the UDDI server.

- 2 Specify the UDDI version.
- 3 Specify the name or part of the name of the service. Select **Exact Match** or **Case Sensitive** to refine your search, if they are applicable. To perform a wildcard search, use the percent (%) character.
- 4 Click **Search**. VuGen displays all of the matching results.
- 5 Double-click on a service in the list to import it.

Choosing a Service from Quality Center

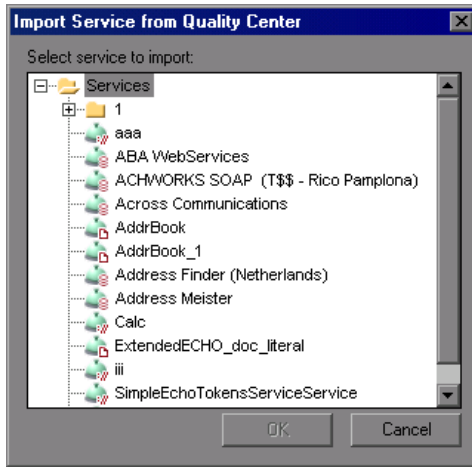
HP Quality Center with the Service Test Management add-on, integrates with VuGen. This integration allows you to store service entries and tests in Quality Center. You can also create and organize services according to your test requirements and test plan.

To specify a service from Quality Center:

- 1 In the Import Service dialog box, select **Quality Center**.
- 2 If you have not yet connected to your Quality Center project, click **Quality Center Connection** to open the Connection dialog box. For information on opening a Quality Center connection, see *Managing Scripts Using Quality Center* in *Volume I-Using VuGen*.
- 3 Click the **Browse** button to view the list of service entries saved in Quality Center.



- 4 Double-click on a service in the list to import it.



Specifying WSDL Connection Settings

VuGen supports the importing of WSDLs using authentication and WSDLs accessed through proxy servers.

Once you enter the security or proxy information, it remains with the WSDL, visible through the **Connection Settings** tab in the Service Management dialog box. If you enable the **Keep up to date** option to allow automatic synchronization, Service Test accesses the WSDL at its source using the authentication or proxy server settings.

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

To specify authentication or proxy information for importing:

- 1 Open the Import Service dialog box as you normally would, either with a new Web Service call, recording, or Traffic Analysis.
- 2 Select either the **URL** or **UDDI** option and specify a URL of the service to be imported.

- 3 In the Import Service dialog box, click the **Connection Settings** button to open the box.

Connection Settings

Authentication

Use Authentication Settings

Username:

Password:

The above values only apply to the imported WSDL. To use these values during replay, add a web_set_user step with the desired values.

Proxy

Use Proxy Settings

Server: Port:

Username:

Password:

The above values only apply to the imported WSDL. To use these values during replay, add a web_set_proxy step with the desired values.

OK Cancel

- 4 In the Connections Settings dialog box, select the desired option: **Use Authentication Settings**, **Use Proxy Settings**, or both.
- 5 Specify the authentication details, and, for a proxy server, the name and port of the server. If you attempt to import the secure service before specifying the necessary credentials, VuGen prompts you to enter the information.
- 6 To update or modify the security settings, open the Service Manager and select the appropriate service in the left pane. Click the **Connection Settings** tab. Edit the required fields and click **OK**.

Deleting Services

You can delete service entries from the Service Management dialog box, when they are no longer required. If a service was updated, you can synchronize the WSDL from the source—you don't need to delete it and reimport the service.

Before deleting a service, make sure that it is not required for your script. If you created a script based on a specific service and you then attempt to delete it, VuGen warns you that the deletion may affect your script. Deletions cannot be undone.

To delete a service, select it from the list of services and click the **Delete** button.

Comparing WSDL Files

When you import a WSDL file, VuGen makes a working copy and saves it along with the script. This saves resources and enables a more scalable and stable environment.

It is possible, however, that by the time you run the script, the original WSDL file will have changed. If you run the script, the test results may be inaccurate and the script may no longer be functional. Therefore, before replaying a Web Services script that was created at an earlier date, you should run a comparison test on the WSDL file.

VuGen provides a comparison tool which compares the local working copy of the WSDL file with the original file on the file system or Web server.

If the differences are significant, you can update the WSDL from the original copy using the **Synchronize** option in the Service Management dialog box.

VuGen also has a general utility that allows you to compare any two XML files. For more information, see "Comparing XML Files" on page 56.

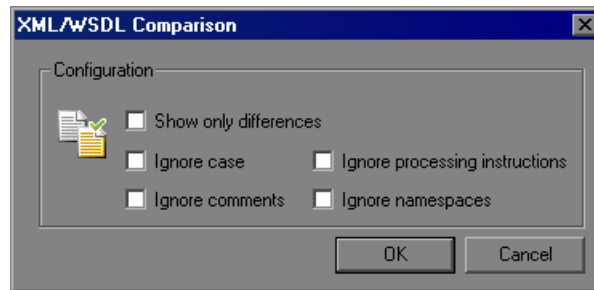
Setting WSDL/XML Comparison Options

VuGen offers the following options when comparing the local and global copies of the WSDL documents, or the revisions of an XML file:

- **Show only differences.** Show only lines with differences. Do not show the entire document.
- **Ignore case.** Ignore case differences between the texts.
- **Ignore comments.** Ignore all comments in the texts.
- **Ignore processing Instructions.** Ignore all texts with processing instructions.
- **Ignore namespaces.** Ignore all namespace differences.

To configure the comparison options:

- 1** Configure the comparison settings. Select **SOA Tools > SOA Settings > XML/WSDL Compare**. The WSDL Operations Options dialog box opens. Select the desired options.



- 2** Click **OK**.

Note: The comparison option settings apply to both WSDL comparisons from within the Service Management window, and for XML comparisons accessed from the **Tools** menu.

Comparison Reports

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

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

In XML Comparison reports, each column displays the path of an XML file.

The Comparison report uses the following legend to mark the differences between the two files:

- ▶ **Yellow.** Changes to an existing element (shown in both versions).
- ▶ **Green.** A new element added (shown in the original file copy).
- ▶ **Pink.** A deleted element (shown in the working copy).

In the following example, line 24 was deleted from the original copy and and line 28 was added.

0:00:10 2005

Found 2 differences.

Working copy	
type>	18
ence>	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
lement name="phone-numbers" type="typens:ArrayOfPhoneNumber"/>	28
ence>	29
type>	30
type>	31

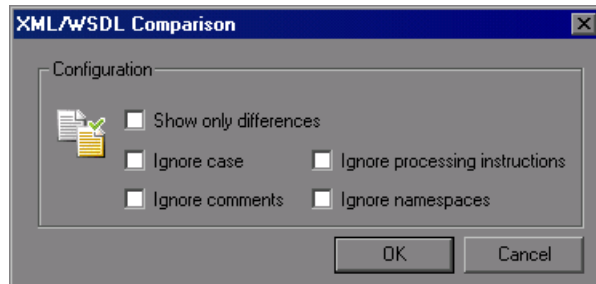
Added line Deleted line

Running a WSDL Comparison

After running a file comparison, you can decide whether to ignore the changes, if they exist, or reload the WSDL file.

To compare WSDL files:

- 1 Configure the comparison settings. Select **SOA Tools > SOA Settings > XML/WSDL Compare**. The XML/WSDL Comparison dialog box opens. Select the desired options.



- 2 Open the Service Management window. Select **SOA Tools > Service Management** or click the **Manage Services** toolbar button
- 3 Select the service upon which you want to perform a comparison. You can only run the comparison on one service at a time.
- 4 Click **Compare**. The WSDL Comparison Report opens.
- 5 Scroll down through the file to locate the differences.

If you find differences between the two files and you want to update VuGen's working copy of the WSDL file, click on the WSDL file in the tree in the left pane. Select **Refresh file from global copy** from the right-click menu. This copies the current version of the WSDL into the script's WSDL directory.

- 6 To close the WSDL Comparison Report window, select **File > Exit**.

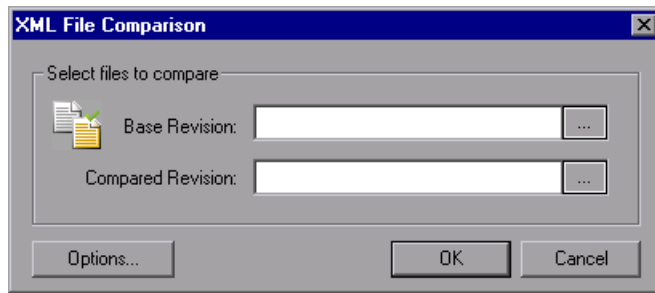
Comparing XML Files

VuGen provides a utility that lets you compare two XML files.

You can specify what differences to ignore, such as case or comments. For additional information about the comparison options, see "Setting WSDL/XML Comparison Options" on page 52.

To compare two XML files:

- 1 Select **Tools > Compare XML Files**. The XML File Comparison dialog box opens.



- 2 Click the Browse button to the right of the **Base Revision** box to locate the original XML file.
- 3 Click the Browse button to the right of the **Compared Revision** box to locate the newer XML file.
- 4 Click **OK**. VuGen opens the XML Comparison Report window.

For information about the Comparison report, see "Comparison Reports" on page 54.

Viewing WSDL Files

The Service Management window lets you view the WSDL in your default browser.

To view a WSDL:

- 1** Select it in the left pane.
- 2** Click the **View WSDL** button in the Service Management window.

4

Web Services - Adding Script Content

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

This chapter includes:

- ▶ About Adding Content to Web Services Scripts on page 60
- ▶ Recording a Web Services Script on page 60
- ▶ Viewing the Workflow on page 65
- ▶ Adding New Web Service Calls on page 67
- ▶ Importing SOAP Requests on page 69
- ▶ Using Your Script on page 72
- ▶ Managing Data in Quality Center on page 73
- ▶ Working with Service Test Management on page 73

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

About Adding Content to Web Services Scripts

Web Services scripts let you test your environment by emulating Web Service clients.

After creating an empty Web Services script, as described in "Creating an Empty Web Services Script" on page 24, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

For information on creating scripts by	See
recording	"Recording a Web Services Script" below
manually inserting Web Service calls	"Adding New Web Service Calls" on page 67
importing SOAP requests	"Importing SOAP Requests" on page 69
analyzing server traffic	Chapter 5, "Web Services - Server Traffic Scripts"

Recording a Web Services Script

By recording a Web Services session, you capture the events of a typical business process. If you have already built a client that interacts with the Web Service, you can record all of the actions that the client performs. The resulting script emulates the operations of your Web Service client. After recording, you can add more Web Service calls and make other enhancements.

Specify Services for Recording

When you record an application, you can record it with or without a Web Service WSDL file. We recommend that you record with a WSDL when possible.

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

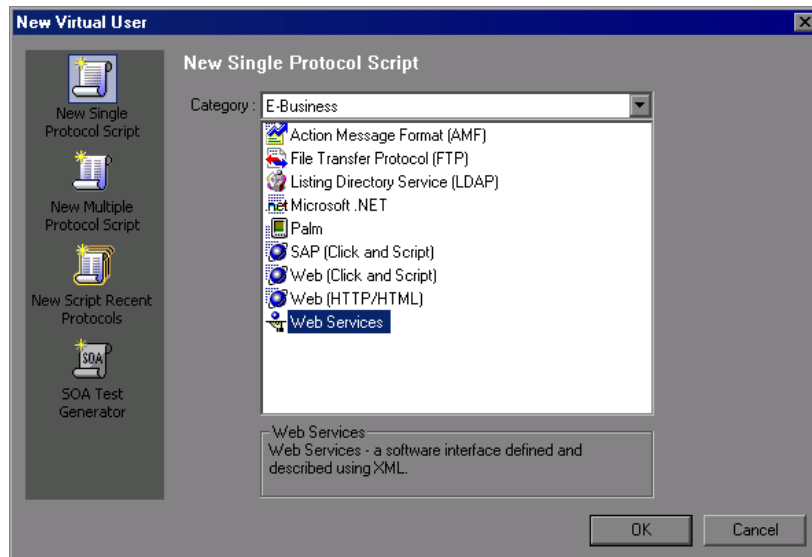
If you do not specify a WSDL file (not recommended), VuGen creates a test with SOAP requests instead of Web Service call steps. If you create a script without importing a service, VuGen creates a `soap_request` step whose arguments are difficult to maintain.

To create Web Services script through recording:

1 Create an empty script.

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

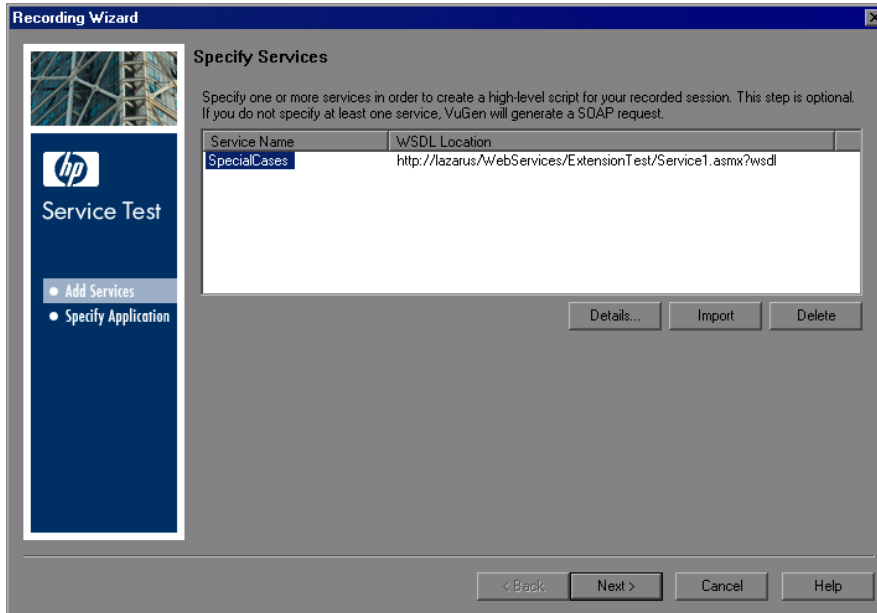
For a single protocol script, click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the E-Business category. Click **OK**.



If you need to record several different protocols, such as Web Services and Web, click **New Multiple Protocol Script** in the left pane and specify the desired protocols. Click **OK**.

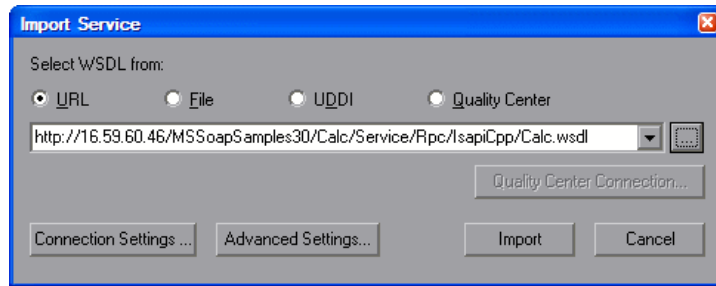
2 Begin the recording process.

Click the **Start Record** button or Ctrl+R to open the Specify Services screen.



3 Add a service to the list.

To produce a high-level Web Service script, add one or more services using the **Import** button. If the WSDL of the recorded Web Service is available, we recommend to import it here. If you create a script without importing a service, VuGen creates a **soap_request** step. The Import Service dialog box opens.



- a** Select a source and location for the WSDL.
- b** Select a toolkit. The toolkit you select is permanently associated with the script. For more information, see "Importing Services" on page 45.
- c** Click **Import**.

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

4 Enter details for the Web Service.

Click **Details** to open the Service Manager window and view the details of the Web Services that you added. For more information, see Chapter 6, "Web Services Call Properties."

In the next step, you select an application to record.

Selecting an Application to Record

In this screen you specify the application to record. You can record a browser session or client application.

Specify application to record

You can choose between the recording of your default browser and the recording of any client application that accesses a Web Service

Select recorded application

Record default web browser

URL:

Record any application

Program to record:

Program arguments:

Working directory:

Options configuration

Record into action:

Record application startup

Advanced Options...

1 Specify the recording details.

- **Record default Web Browser.** Records the actions of the default Web browser, beginning with the specified URL. Select this option where you access the Web Service through a Web-based UI.
- **Record any application.** Records the actions of a your client application. Specify or browse for the path in the **Program to record** box. Specify any relevant arguments and working directories.

2 Configure options.

- **Record into action.** The action in which to generate the code. If there are startup procedures that you do not need to repeat, place them in the **vuser_init** section. During recording you can switch to another section, such as **Action**.

- **Record application startup.** Records the application startup as part of the script. If you want to begin recording at a specific point, not including the startup, clear this check box.
- **Advanced Options.** Opens the Recording Options dialog box, allowing you to customize the way in which the script is generated. For more information, see the section on setting Script Generation preferences.

3 Click Finish.

VuGen opens the application and begins recording. Perform the desired actions within your application and then press the **Stop** button on the floating toolbar. VuGen generates **web_service_call** functions, or **soap_request** functions if you did not import a WSDL.

After recording, you can enhance your script with additional service calls and parameterization. You run the completed script in VuGen to check its functionality. You run the completed script in VuGen to check its functionality.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 22.

Viewing the Workflow

When adding script content through recording or Analyzing traffic, VuGen's workflow guides you through the stages of preparing a script. By clicking in the **Tasks** pane, you can read about the steps for creating a script, view information about your recording, and verify the replay. Use the **Back** and **Next** buttons to navigate between screens.

If you do not see the Workflow screen, click the **Tasks** button on the toolbar to open the Tasks pane, and select a task.

Create Script

The Create Script screen provides several guidelines for creating a Web Services script. It also provides a link for opening the Web Services wizard.

- ▶ **Before You Start.** Describes what you should know before you begin.
- ▶ **About Script Creation.** Describes the stages of script creation.
- ▶ **Actions.** Describes script sections and why they are important.

There are two action-related links:

- ▶ **Start Recording.** Opens the Start Recording dialog box where you provide information about the application to record.
- ▶ **Analyze Traffic.** Lets you analyze traffic captured over the network to create a script that emulates a server.

Creation Summary

After you create a script, the Creation Summary screen provides information about the recording or script generation.

- ▶ **Protocols.** Lists which protocols were used during the script creation.
- ▶ **Actions.** Describes into which sections actions were recorded or imported.

It also provides links that allow you to modify the script:

- ▶ **Add a web_service_call statement.** Lets you manually add a `web_service_call` function to your script by specifying a service, operation, and argument values.
- ▶ **Manage the services.** Opens the Service Repository window with a list of all of the services that are available to the script and their properties.
- ▶ **Compare XML files.** Allows you to compare two versions of an XML file. This is useful for comparing WSDL files and determining if there was a change since your originally imported it into the script.

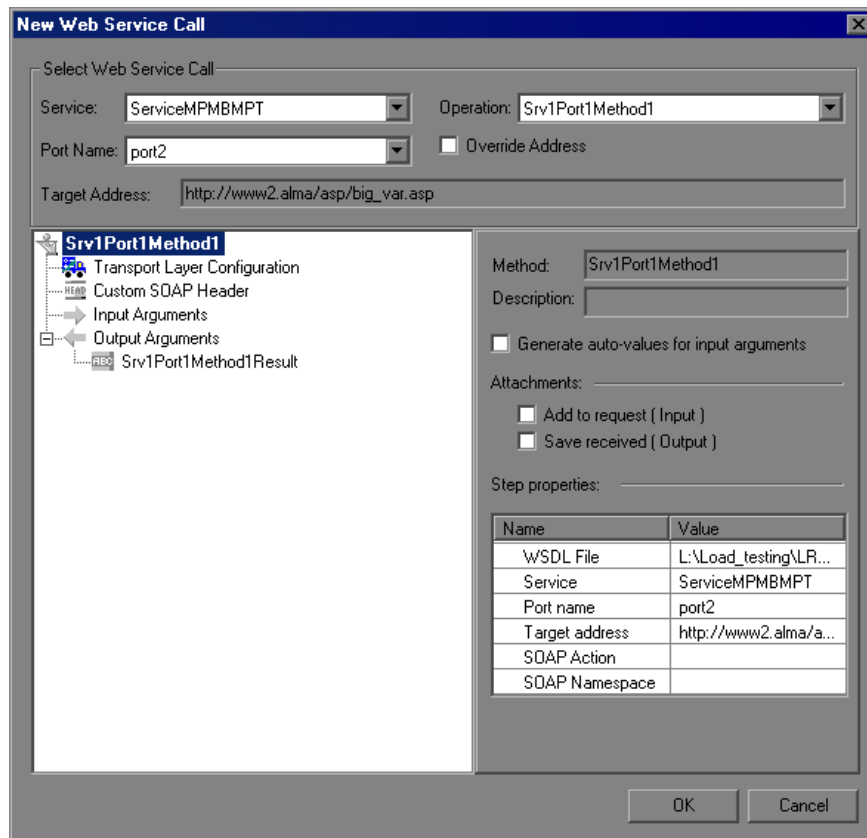
For information about the remainder of the workflow, see Chapter 4, "Viewing the VuGen Workflow."

Adding New Web Service Calls

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

To add a Web Service call:

- 1 Click the cursor at the desired location in your script.
- 2 Click the **Add Service Call** button. The New Web Service Call dialog box opens. Select the desired **Service**. If this is a new script and you did not yet import a WSDL, do so at this point. For more information, see "Importing Services" on page 45.
- 3 Select an **Operation**. For services using multiple ports, select a **Port Name** for the operation. This lets you differentiate between identical operations performed on separate ports.



- 4** To specify a target address other than the default for the active port, select **Override address** and enter the address.
- 5** To provide sample input values for the service, click on the highest level node (the service name) and select **Generate auto-values for input arguments**. VuGen adds sample values and places an arrow before each of the arguments, to indicate that it is using auto-values.

To provide sample values for all Input arguments, select the **Input Arguments** node and click **Generate**.
- 6** To parameterize the input arguments of the operation, see Chapter 14, "File, Table, and XML Parameter Types" in *Volume I-Using VuGen*.
- 7** Select the **Transport Layer Configuration** node to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see Chapter 11, "Web Services - Transport Layers and Customizations."
- 8** Click on each of the nodes and specify your preferences for argument values. For more information, see Chapter 6, "Web Services Call Properties."
- 9** To add an attachment to an input argument, or to specify a parameter to store output arguments, select the operation's main node and make the appropriate selection. For more information, see "Attachments" on page 113.

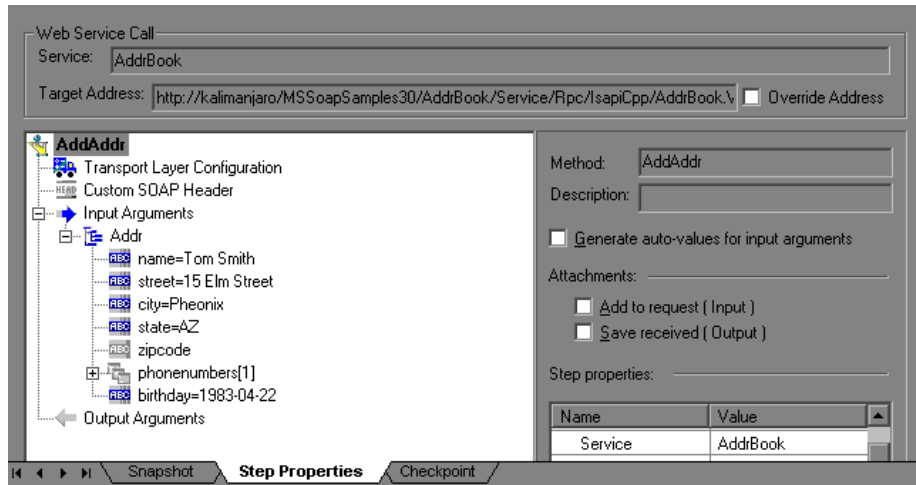
Importing SOAP Requests

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

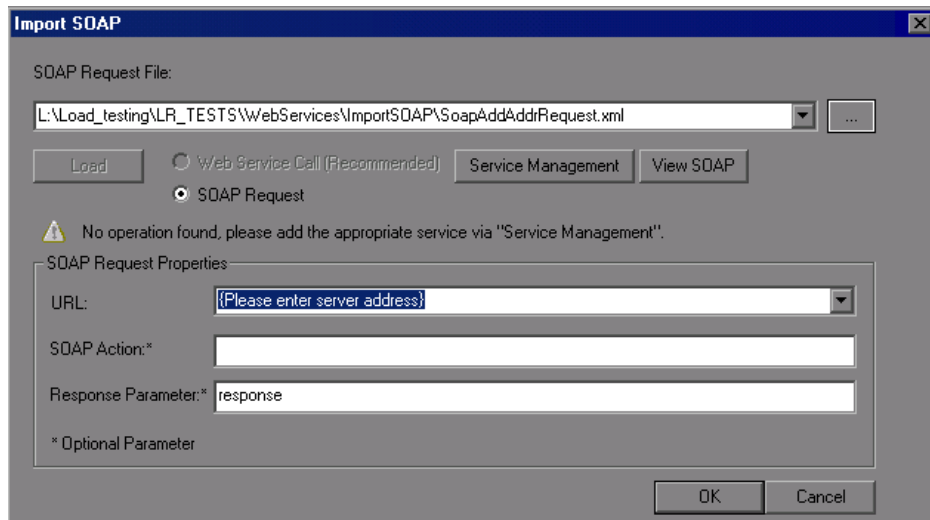
For example, suppose you have a SOAP request with the following elements:

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

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



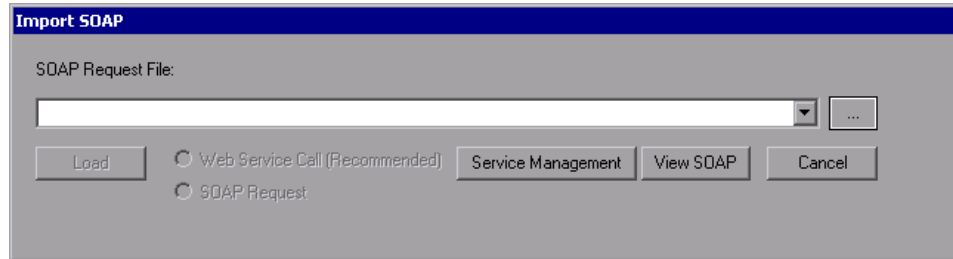
To create a new Web Service call based on a SOAP request, you must first import a WSDL file. If a WSDL is not available, or if you want to send the SOAP traffic directly, you can create a SOAP Request step. You specify the URL of the server, the SOAP action, and the response parameter.



In Script view, the SOAP Request step appears as a `soap_request` function, described in the *Online Function Reference* (**Help > Function Reference**).

To import a SOAP request:

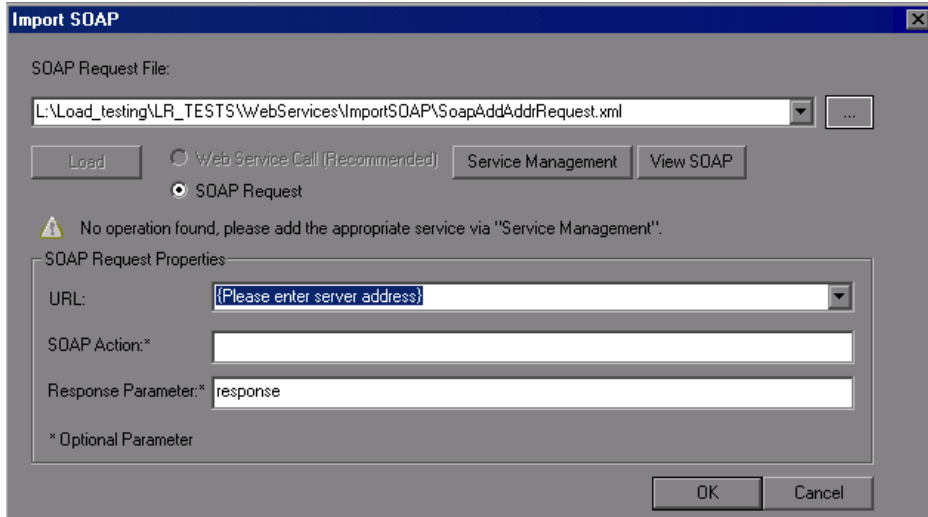
- 1 Click the **Import SOAP** button or select **SOA Tools > SOAP Import**.



- 2 Browse for the XML file that represents your SOAP request.
- 3 Select the type of step you would like to generate: **Create Web Service Call** or **Create SOAP Request**. In order to create a Web Service Call, you must first import at least one WSDL that describes the operation in the SOAP request file. To import a WSDL, click **Service Management** and then click the **Import** button. To view the SOAP before loading it, click **View SOAP**.
- 4 Click **Load**. VuGen loads the XML element values.

For a Web Service Call, set the properties for the Service call as described in "Understanding Web Service Call Properties" on page 93.

For a SOAP request, provide the URL and the other relevant parameters.



- 5 For a Web Service Call, if there are multiple services with same operation (method) names, you need to select the service whose SOAP traffic you want to import. For information about additional properties, see "Understanding Web Service Call Properties" on page 93.
- 6 Click **OK** to generate the new step within your script.
- 7 Set checkpoints and replay the step. For more information, see "Checkpoints" on page 122.

Using Your Script

After you create scripts, you can manage them in one of the following ways:

- ▶ **Service Test Management.** An add-on for HP Quality Center that lets you manage SOA testing by allowing you to import, store and define services in Quality Center. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management. For more information, see "Working with Service Test Management" on page 73.

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 7, "Web Services - Preparing for Replay."
- ▶ **Load Testing.** Integrate the script into a LoadRunner Controller scenario to test its performance under load. For more information, see the *HP LoadRunner Controller* or *Performance Center* documentation.
- ▶ **Production Testing.** Check your Web service's performance over time through a Business Process Monitor profile. For more information, see the *HP Business Availability Center* documentation.

Managing Data in Quality Center

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

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

For more information, see the *HP Quality Center User Guide*.

Working with Service Test Management

HP Quality Center is a Web-based application for test management. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management.

The **Service Test Management** add-on for Quality Center, lets you manage SOA testing by allowing you to import, store and define services in Quality Center.

The Service Test Management integration lets you:

- ▶ **Store Web Services.** You can store and organize Web Services in Quality Center for use within Service Test.
- ▶ **Write Service Test scripts.** You can create scripts based on the services stored in Quality Center, while maintaining up-to-date WSDLs throughout the life-cycle of the service and the script.
- ▶ **Compose a Business Process Test.** You can create a BPT (Business Process Test) in Quality Center based on services defined through Service Test Management.

Service Test Management also integrates with HP's Systinet Registry, to create test requirements and plans. Once the services are imported, Service Test Management identifies any changes to the services and automatically generates the necessary test cases that need to be run.

Using the **Service Test Management** add-in for Quality Center, groups throughout your organization can contribute to the quality process in the following ways:

- ▶ Business analysts define application requirements and testing objectives.
- ▶ Test managers and project leads design test plans and develop test cases.
- ▶ Test managers automatically create QA testing requirements and test assets for SOA services and environments.
- ▶ Test automation engineers create automated scripts and store them in the repository.
- ▶ QA testers run manual and automated tests, report execution results, and enter defects.
- ▶ Developers review and fix defects logged into the database.
- ▶ Project managers can export test and resource data in various reports, or in native Microsoft Excel for analysis.
- ▶ Product managers decide whether an application is ready to be released.
- ▶ QA analysts can auto-generate test asset documentation in Microsoft Word format.

For more information about the integration, see the *HP Service Test Management User Guide*.

5

Web Services - Server Traffic Scripts

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

This chapter includes:

- ▶ About Creating Server Traffic Scripts on page 77
- ▶ Getting Started with Server Traffic Scripts on page 79
- ▶ Generating a Capture File on page 80
- ▶ Creating a Basic Script from Server Traffic on page 82
- ▶ Specifying Traffic Information on page 83
- ▶ Choosing an Incoming or Outgoing Filter on page 84
- ▶ Providing an SSL Certificate on page 86

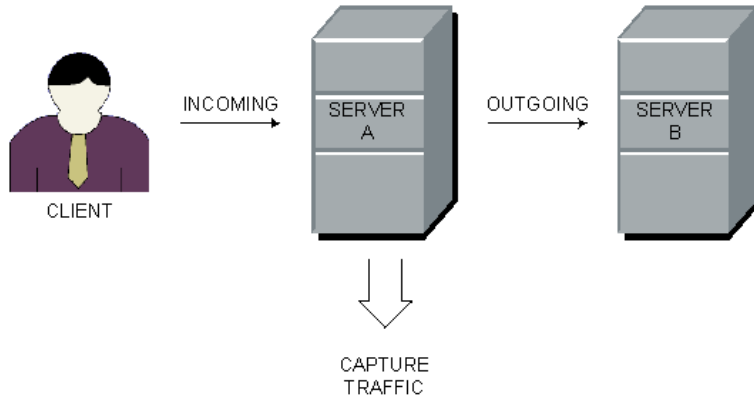
The following information only applies to Web Services/SOA Vuser scripts.

About Creating Server Traffic Scripts

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

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

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server. The steps in creating a script by analyzing server traffic are described below in Getting Started with Server Traffic Scripts.



There are two types of emulations: **Incoming traffic** and **Outgoing traffic**.

Incoming traffic scripts emulate situations in which you want to send requests to the server, but you do not have access to the client application, for example, due to security constraints. The most accurate solution in this case is to generate a script from the traffic going **into** the server, from the side of the client.

When you specify an Incoming server network traffic, you indicate the IP address of the server and the port number upon which the application is running. VuGen examines all of the traffic going into the server, extracts the relevant messages, and creates a script. In the above diagram, if the client is unavailable, you could create an Incoming script to emulate the requests coming into **Server A**.

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

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

Getting Started with Server Traffic Scripts

The following section outlines the process of creating a script that analyzes server traffic.

1 Create a capture file.

VuGen uses the capture file to analyze the server traffic and emulate it. For more information, see "Generating a Capture File" on page 80.

2 Create a new Web Services script.

Using VuGen's main interface, you create a new Web Services script. For more information see Chapter 4, "Web Services - Adding Script Content."

3 Specify the Services (optional, but recommended).

To create a high-level script, import a WSDL which describes the Web Service you want to test.

4 Specify the traffic information.

Click the **Analyze Traffic** button. Specify the location of the traffic file and whether your script will be for Incoming or Outgoing traffic. For more information, see "Specifying Traffic Information" on page 83.

5 Specify the traffic filter Recording options.

Filter options let you determine which hosts to include or exclude in your script. For more information, see "Choosing an Incoming or Outgoing Filter" on page 84.

6 Specify the SSL certificate information.

The SSL configuration lets you analyze secure traffic over HTTPS in order to generate the script. For more information, see "Providing an SSL Certificate" on page 86.

Generating a Capture File

A capture file is a trace file containing a log of all TCP traffic over the network. Using a sniffer application, you obtain a dump of all of the network traffic. The sniffer captures all of the events on the network and saves them to a capture file.

To generate a smaller, more manageable script, try to capture the network traffic only for the time that you perform actions in your application.

Note: Capture files do not contain loopback network traffic.

You can obtain a capture file using the command line utility or any existing capture tool.

The Capture File Command Line Utility

The VuGen command line utility, **lrtcpdump**, is located in the product's **bin** directory. There is a separate utility for each of the platforms: **lrtcpdump.exe** (Windows), **lrtcpdump.hp9**, **lrtcpdump.ibm**, **lrtcpdump.linux**, and **lrtcpdump.solv4**.

To invoke the capture tool, type:

```
lrtcpdump -i<interface> -f<file>
```

where **interface** is the name of the network card whose traffic you want to capture, and **file** is the name of the capture file in which to store the information. Do not leave a space between the command line option (**i** or **f**) and the value.

To create a capture file on a Windows platform:

- 1 Select **Start > Run**, type **cmd** and click **OK** to open a command window.
- 2 Drag in or enter the full path of the **lrtcpdump.exe** program located in the product's **bin** directory.
- 3 Provide a file name for the capture file using the following syntax:

```
lrtcpdump -f <file>
```


for example **lrtcpdump -fmydump.cap**.

- 4** **lrtcpdump** prompts you to select a network card. If there are multiple interface cards, it lists all of them. Type in the number of the interface card (1, 2, 3 etc.) and click **Enter**.
- 5** Perform typical actions within your application.
- 6** Return to the command window and click **Enter** to end the capture session.

To create a capture file on a UNIX platform:

- 1** Locate the appropriate **lrtcpdump** utility for your platform in the product's **bin** directory. Copy it to a folder that is accessible to your UNIX machine. For example, for an HP platform, copy **lrtcpdump.hp9**. If using FTP, make sure to use the binary transfer mode.
- 2** Switch to the root user to run the utility.
- 3** Provide execution permissions. **chmod 755 lrtcpdump.<platform>**
- 4** On UNIX platforms, if there are multiple interface cards, **lrtcpdump** uses the first one in alphabetical order. To get a complete list of the interfaces, use the **ifconfig** command.
- 5** Run the utility with its complete syntax, specifying the interface and file name. For example, **lrtcpdump.hp9 -ietho -fmydump.cap**. The capturing of the network traffic begins.
- 6** Perform typical actions within your application.
- 7** Return to the window running **lrtcpdump** and follow the instructions on the screen to end the capture session.
- 8** Place the capture file on the network in a location accessible to the machine running VuGen.

An Existing Capture Tool

Most UNIX operating systems have a built-in version of a capture tool. In addition, there are many downloadable capture tools such as **Ethereal/tcpdump**.

When using external tools, make sure that all packet data is being captured and none of it is being truncated.

Note: Certain utilities require additional arguments. For example, `tcpdump` requires the `-s 0` argument in order to capture the packets without truncating their data.

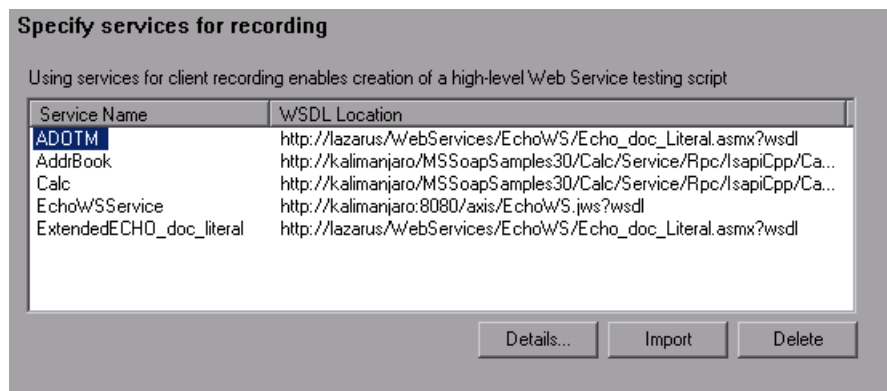
Creating a Basic Script from Server Traffic

You create a script from server traffic just as you would create a recorded script.

You can optionally specify a Web Service for your script. If you specify a service, VuGen will create a script with `web_service_call` functions. If you do not specify a service, VuGen creates a script with `soap_request` functions.

To create a server traffic script:

- 1** Select **File > New** and click **New Single Protocol Script** in the left pane.
- 2** Select the **Web Services** protocol and click **OK**.
- 3** Click the **Analyze Traffic** button or select **Vuser > Analyze Traffic**. The wizard opens.
- 4** Add one or more services to the list. This step is optional.



- To add a new service, click **Import**. In the Import Service dialog box, specify the location of the WSDL. You can specify a URL, File, UDDI server (such as Systinet), or a location in Quality Center. In the Import Service dialog box, you also select a toolkit for analyzing the service. The selected toolkit will be permanently associated with the script—it cannot be changed. For more information, see "Importing Services" on page 45.
 - To set or view details about the services, click **Details** to open the Service Management window. For more information about Service Management, see Chapter 3, "Web Services - Managing."
 - To remove a listed service, select it and click **Delete**
- 5 On the bottom of the wizard screen, click **Next** to specify the traffic file information. For more information, see below.
 - 6 After providing traffic information, click **Finish** to generate a script.

Specifying Traffic Information

The traffic file contains a dump of all the network traffic. Using the wizard, you specify the location of the traffic file and whether you want your script to emulate incoming or outgoing traffic.

Specify traffic information

Traffic Data

Capture file: [] [...]

Incoming traffic
 Server: [] Port: []

Outgoing traffic
 Server: []

Options Configuration

Record into action: Action []

[Filter options....] [SSL Configuration...]

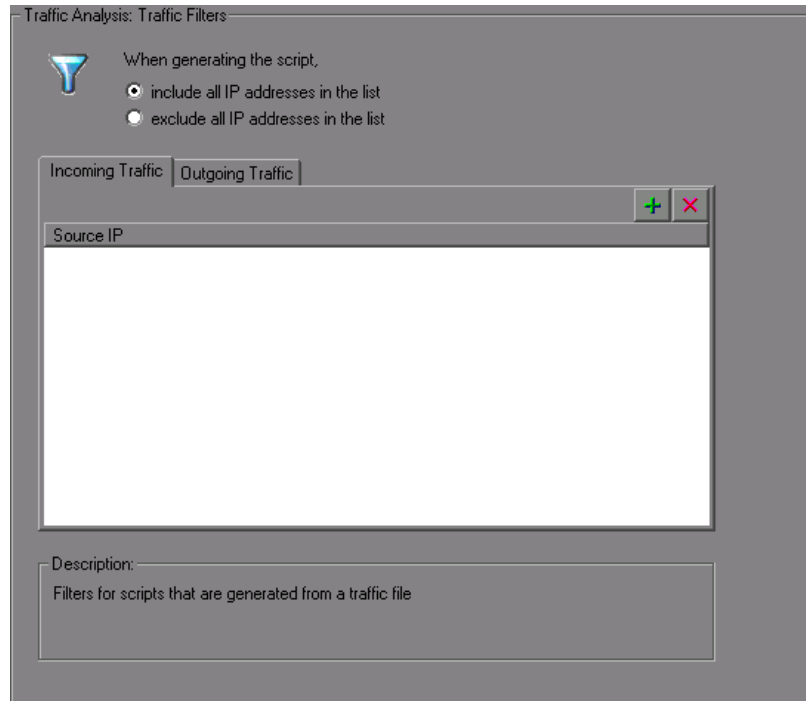
- ▶ **Capture file.** The name and path of the traffic file, usually with a cap extension.
- ▶ **Incoming traffic:Server/Port.** The IP address and port of the server whose incoming traffic you want to examine.
- ▶ **Outgoing traffic:Server.** The IP address of the server whose outgoing traffic you want to examine.
- ▶ **Record into action.** The section into which to create the script. If you want to use iterations, specify the **Actions** section.
- ▶ **Filter options.** Opens a filter interface allowing you to specify which IP addresses to include or exclude from the script. For more information, see below.
- ▶ **SSL Configuration.** Allows you to add SSL certificates to analyze traffic from a secure server with the required credentials. For more information, see "Providing an SSL Certificate" on page 86.

Choosing an Incoming or Outgoing Filter

You can provide a filter to drill down on specific requests going to or from a server, by specifying its IP address and port.

It is also possible to filter your capture file with an external tool before loading it into VuGen. In that case, you may not require additional filtering.

You filter the requests by choosing the relevant host IP addresses. The filter can be inclusive or exclusive—you can include only those IPs in the list, or include everything except for those IPs that appear in the list.



To filter the traffic file:

- 1** Open the Traffic Filters recording options. Click **Filter Options** in the Specify Traffic Information step, or select **Tools > Recording Options**. Select the **Traffic Analysis:Traffic Filters** node.
- 2** Select one of the filtering options: **include all IP addresses in the list** or **exclude all IP addresses in the list**.
- 3** Select the tab that corresponds to your script type: **Incoming Traffic** or **Outgoing Traffic**.
- 4** Add hosts to the list.



To add a host to the list, click the **Add** button. Specify the IP address of the server you want to add to the list. For incoming traffic, specify the port of the server to include or exclude. Click **OK** to accept the settings.



Click **Delete** to remove an entry.

After the script is created you can change the filters and regenerate the script—there is no need to re-analyze the capture file.

Providing an SSL Certificate

To analyze traffic from a secure server, you must provide a certificate containing the private key of the server.

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

To specify an SSL certificate:

- 1** In the Specify Traffic Information screen, click **SSL Configuration**.
- 2** Add certificates to the list.



To add a certificate to the list, click the **Add** button. Specify the IP address, port, path of the certificate file (with a **pem** extension), and the password for the certificate. Make sure the **pem** file contains the private key. If you are unsure of how to obtain the certificate, contact your system administrator.

IP	Port	File	Password
255.34.0.0	7070	serv512.pem	*****



- Click the **Delete** button to remove an entry from the list.
- 3** Repeat the above steps for every certificate you want to add.
- 4** Click **OK** to close the dialog box.

6

Web Services Call Properties

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

This chapter includes:

- ▶ About the Web Service Call View on page 89
- ▶ Viewing Web Services SOAP Snapshots on page 90
- ▶ Understanding Web Service Call Properties on page 93
- ▶ Derived Types on page 103
- ▶ Working with Optional Parameters on page 104
- ▶ Base 64 Encoding on page 108
- ▶ Attachments on page 113
- ▶ Working with the XML on page 117

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

About the Web Service Call View

Using the Web Service Call view, you can view snapshots, set properties, and define checkpoints for each of the Web Service calls in your script.

To open the Web Service Call view, you must be in Tree view. Select **View > Tree View** in the VuGen window to open Tree view.

The Snapshot lets you view the SOAP structure of each step. You can view snapshots of the recording, replay, request, and response. For more information, see below.

The Properties tab lets you configure the settings for each Web Service call. You can set properties in the area of argument values, parameterization, transport layer, and security. For more information, see "Understanding Web Service Call Properties" on page 93.

Checkpoints let you verify your Web service results. You can check for expected values and view the output to see if they were matched. For more information, see "Checkpoints" on page 122.

Viewing Web Services SOAP Snapshots

You can use VuGen's snapshot viewer to examine the SOAP requests and responses that occurred during record and replay. Note that you must replay the session at least once in order to view a replay snapshot.

There are several ways to view the SOAP snapshots:

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

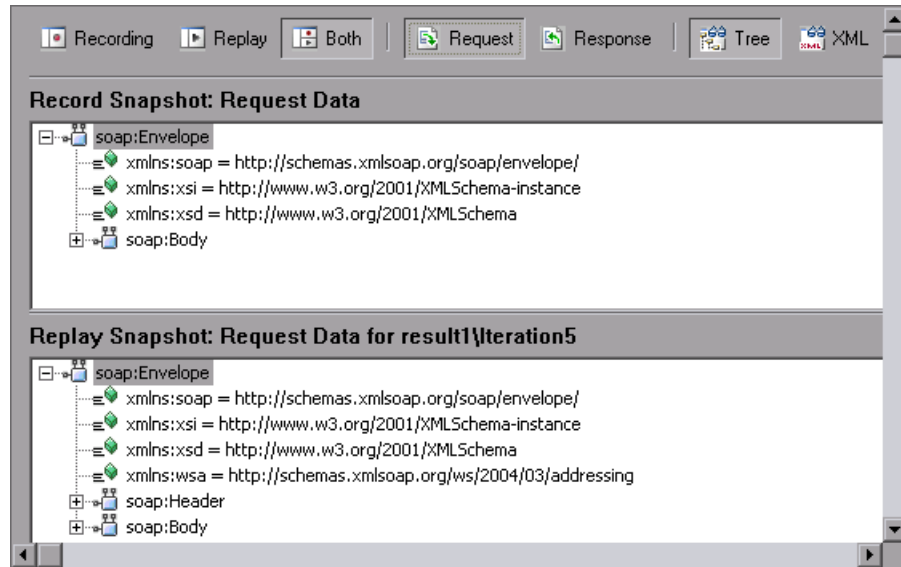
Using the buttons in the Snapshot window, you can control the view:



In **Tree** view, you can expand the nodes to view the values of the arguments, if they were assigned.

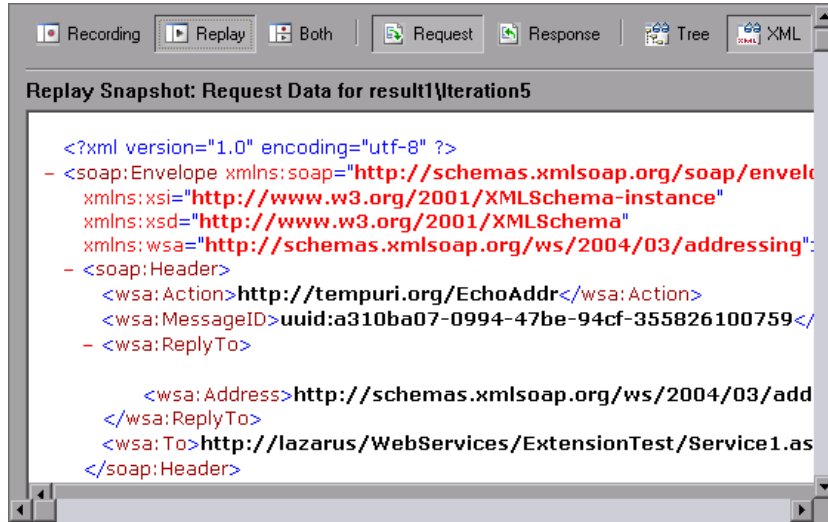
In **Request** view, the displayed values are those that were sent to the server during the Web Services session. In **Response** view, the displayed values are the results returned by the server.

In the following example, the Snapshot window shows the **Record** and **Replay** snapshots of the **Request** data in **Tree** view.



To learn more about a node, select it and select **Node Properties** from the right-click menu.

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



Specifying a Replay Iteration

If you replayed the script with multiple iterations, you can specify which iteration to display in the snapshot. In addition, you can display a snapshot from test results that you saved in a location other than the script's folder. By default, the Snapshot view shows the last iteration.

To specify an iteration to display:

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

Understanding Web Service Call Properties

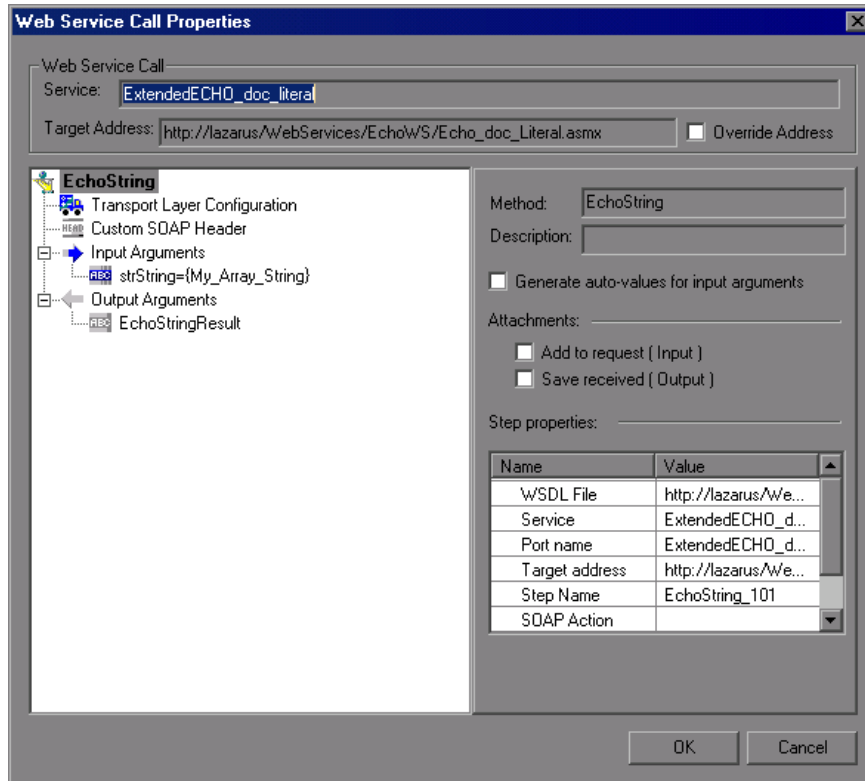
VuGen provides an interface for you to view and modify the properties of each one of the Web Service calls.

Properties describe the behavior of each method within your service. You can set a target address, argument values, parameterization, and transport layer preferences for each of your service's methods.

You can view a step's properties from Tree view (**View > Tree view**) in one of the following ways:

- ▶ Double-click on a step in the left pane to open the Web Service Call Properties dialog box.
- ▶ Select the **Step Properties** tab in the right pane.

The Properties view displays each of the service’s operations in a tree hierarchy. The nodes of the tree represent the Transport Layer Configuration, the SOAP header, input arguments, and output arguments.



By default, the script takes the target address from the WSDL. You can override this address for each operation. Select the **Override Address** option and specify the desired **Target Address**.

The contents of the right pane changes, depending on the level of the selected tree node. The following table describes the content for each node:

If you select...	The right pane shows...
A method or operation name	<ul style="list-style-type: none"> ▶ A checkbox to include input/output attachments ▶ The step properties

Transport Layer Configuration	<p>Advanced transport options:</p> <ul style="list-style-type: none"> ➤ HTTP/S Transport with Async or WSA support. ➤ JMS Transport support with response and request queue information.
SOAP Header	<p>An edit box to indicate the value of the SOAP header for the current element. For more information, see "SOAP Headers" on page 117.</p>
Input Arguments node	<ul style="list-style-type: none"> ➤ The Name of each method and its Value. ➤ Include All, Reset and Generate buttons.
An individual argument	<ul style="list-style-type: none"> ➤ Name. The name of the argument (read-only) ➤ Type. The argument type as defined in the WSDL. When the WSDL contains derived types, this box becomes a drop down list. For more information, see "Derived Types" on page 103. ➤ Include argument in call. Includes the Optional parameters in the Web Service call. See "Working with Optional Parameters" on page 104. ➤ Nil. Sets the Nillable attribute to True. ➤ Value. The value of the argument. For base64 binary type arguments, Get from file or Base64 Encoded text. ➤ Generate auto-value for this argument. Insert automatic values for this node.
A complex input argument	<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. To import SOAP, see "Importing SOAP Requests" on page 69. ➤ Generate auto-value for this argument. Inserts automatic values for all arguments of this complex type node. ➤ Add/Delete. Adds or removes elements from the array.

Output Arguments	<ul style="list-style-type: none"> ➤ The output argument list ➤ Negative Testing options, as described in Chapter 13, "Web Services - Negative Testing"
Input Attachments	The Attachment Format for encoding the soap request: DIME or MIME for the VuGen toolkit, DIME only for .NET (see "Including MIME Attachments" on page 222), and MIME only for Axis.
Attachment	<ul style="list-style-type: none"> ➤ Take Data from. The name of the file to attach or the name of the parameter containing the data. ➤ Content Type. The attachment's content type. You can instruct VuGen to detect it automatically or select a type from the dropdown list or enter a value manually. ➤ Content ID. The attachment's ID attribute. You can instruct VuGen to automatically generate a value or specify your own ID. ➤ Delete Attachment. A button to remove the attachment.
Output Attachments	<ul style="list-style-type: none"> ➤ Save All Attachments. Saves all attachments to parameters: ➤ Content. The name of the parameter prefix for storing the attachment. ➤ Content Type. The attachments' content type attribute (read-only). ➤ Content ID. The attachment's ID attribute (read-only). ➤ Save Attachments by Index. Save the attachments by number, beginning with 1. Click Add to insert additional attachment indexes.

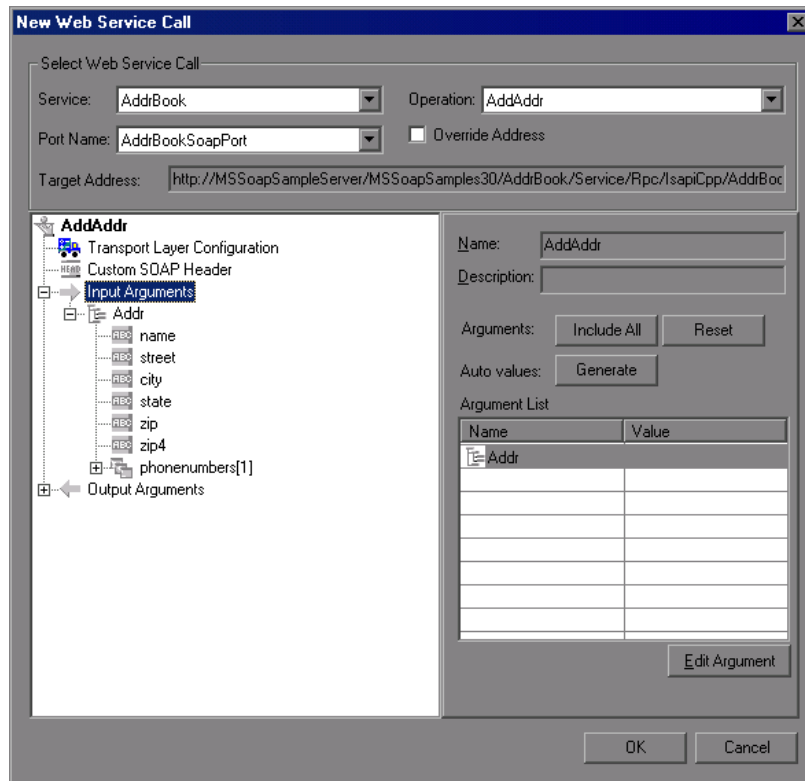
You can set argument values for the following elements: (manually edited arguments are displayed in blue)

- Input Argument Values
- Output Arguments
- Arrays

- Attachments
- SOAP Headers

Input Argument Values

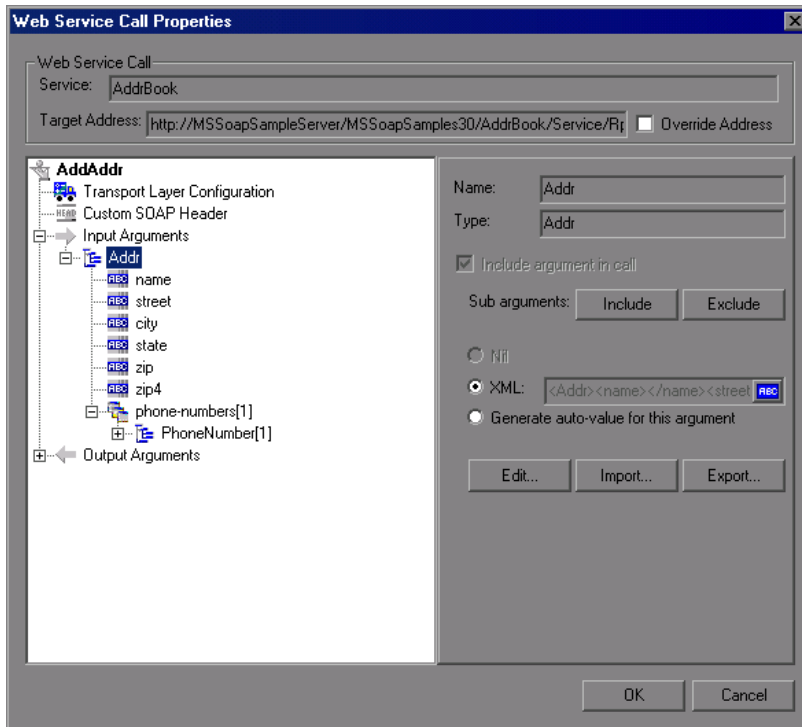
The Input Arguments node lets you define values for all of the input arguments and lets you control the Optional elements. For more information about Optional elements, see "Working with Optional Parameters" on page 104.



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

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

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

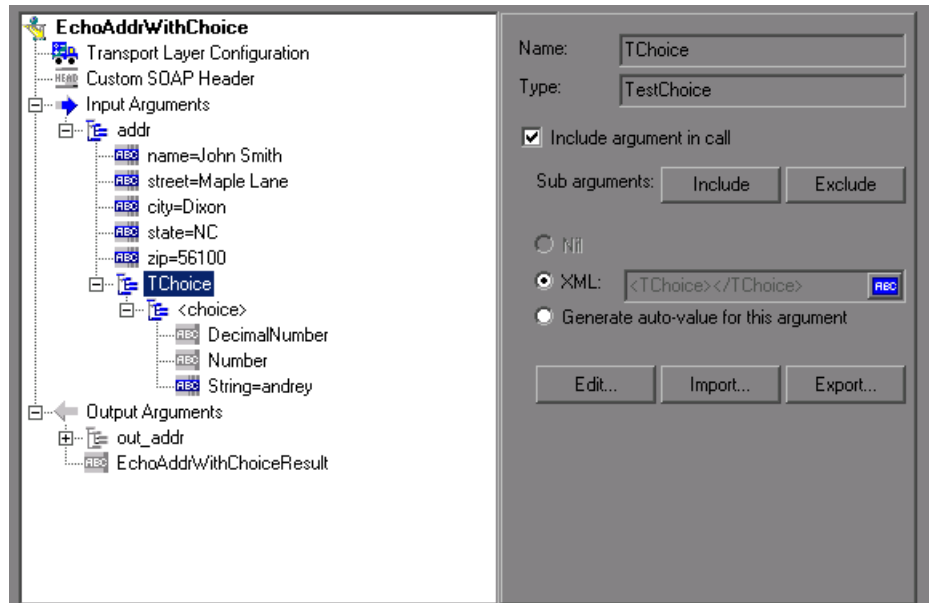


- ▶ **XML/Value.** A manually specified value for the node. If your argument is an array, you can specify an XML structure. Otherwise, specify an ordinary value. To create a parameter for the argument, click the **ABC** icon in the right corner of the **XML/Value** box to open the Select or Create Parameter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

- **Generate auto-value for this argument.** If you want VuGen to automatically generate a value for this argument, select this option or select the argument in the tree hierarchy and select **Generate Auto-values** from the right-click menu.

Choice Elements

If your WSDL defines Choice elements, you can view them and set their values in the Properties view.



To set a value for a choice element, select the parent element, enable the **Include argument in call** option in the right pane, and provide a value.

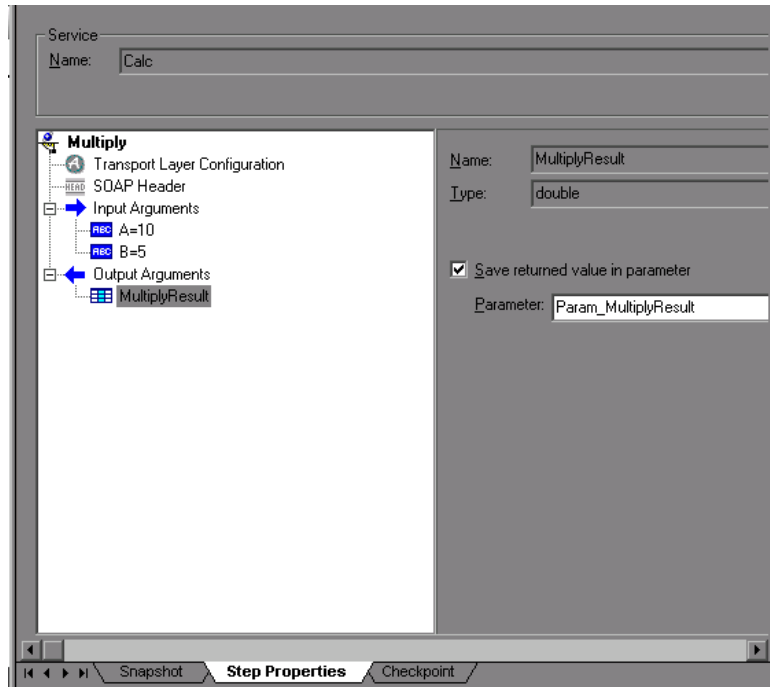
To parameterize the argument, click the **ABC** icon. In the Parameter Properties dialog box, provide values for the choice argument. You only need to provide values for one of the choice elements. When running multiple iterations, the script uses the values for the same choice element, according to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

Choice support is provided for both Input and Output arguments, Parameterization, Checkpoints, and Service Emulation.

For information about working with optional arguments, see "Working with Optional Parameters" on page 104.

Output Arguments

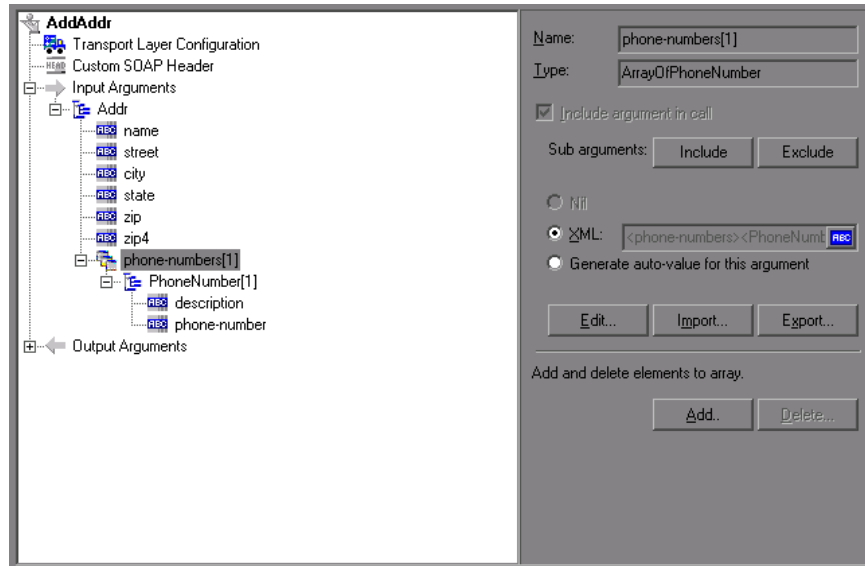
You can view the output argument values and save them to parameters or in an array.



- **Save returned value in parameter.** Saves the returned value to a parameter whose name you specify in the text box.

Arrays

To work with an array—for either input or output arguments, select it in the left pane.



- **XML.** The path of the XML file containing the values of the array elements. Click the **ABC** icon to replace the XML with an XML type parameter. XML parameterization supports arrays as input arguments. In the XML parameter, you define the number of array elements as required.

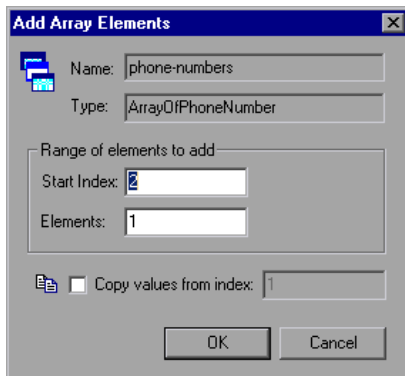
When saving an array to a parameter, the number of array elements per parameter is constant. If you want to run multiple iterations, with each iteration using a different number of array elements, you need to define separate parameters, each containing the desired number of array elements.

For more information about XML parameters, see "Setting Properties for XML Parameters" in *Volume I-Using VuGen*.

- ▶ **Edit/Import/Export.** To modify complex types and arrays, select the elements and arguments and click **Edit**. Click **Export** to export the selected entry to a separate XML file, or **Import** to load a previously exported XML file. **Note:** This **Import** operation handles XML files that were previously exported—not standard SOAP files. To import SOAP, see "Importing SOAP Requests" on page 69.
- ▶ **Add.** Opens the Add Array Elements dialog box, allowing you to add array elements, either simple or complex.
- ▶ **Delete.** Deletes array elements. Specify the starting index and the number of elements to remove.

Adding Array Elements

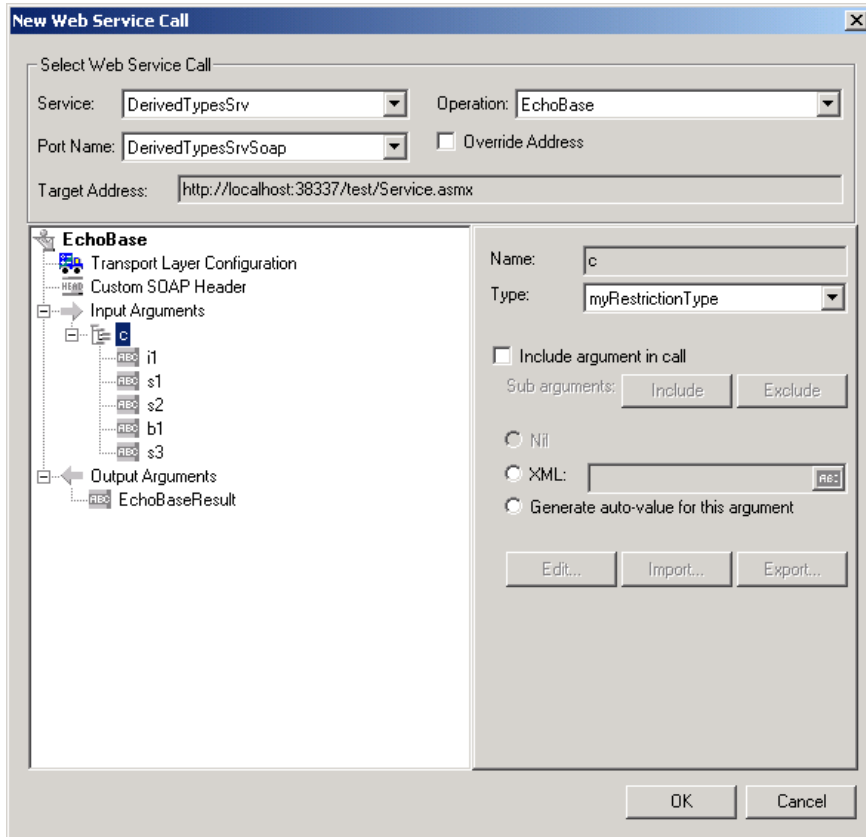
When you click **Add** in the Array Elements section, the Add Array Elements dialog box opens as described below.



- ▶ **Start Index.** The index of the first element that will be added.
- ▶ **Elements.** The number of elements to add.
- ▶ **Copy values from index.** Assigns values of an existing array element to the new elements. Specify the array index of the element whose value you want to use.

Derived Types

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



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

Abstract Types

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

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

Working with Optional Parameters

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

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

`minoccurs='0'`

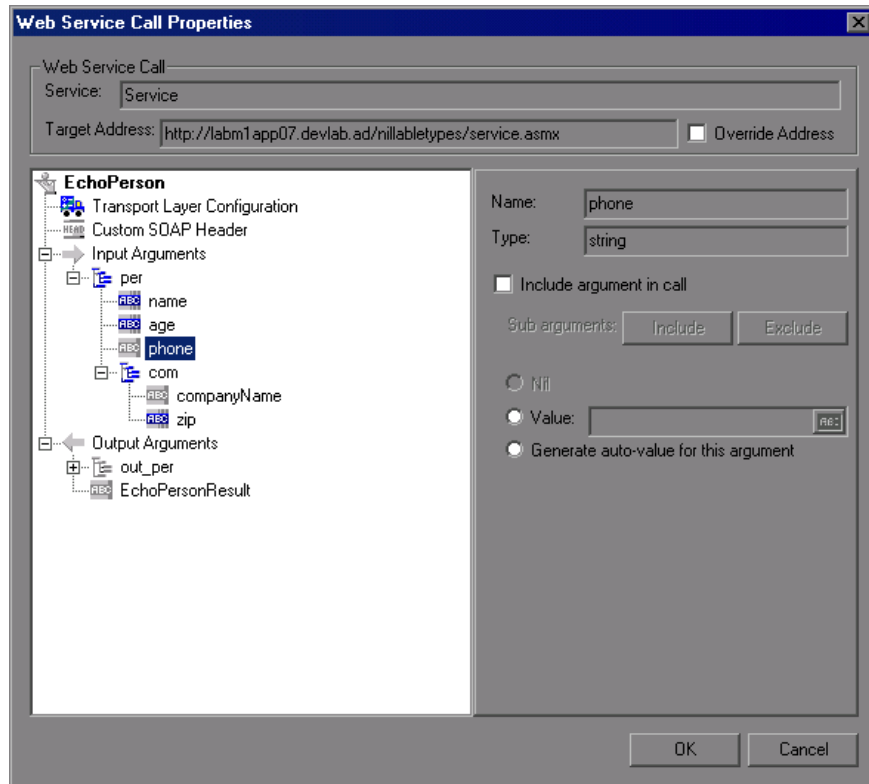
`nillable='true'`

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

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

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


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



The following table indicates the availability of the options:

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

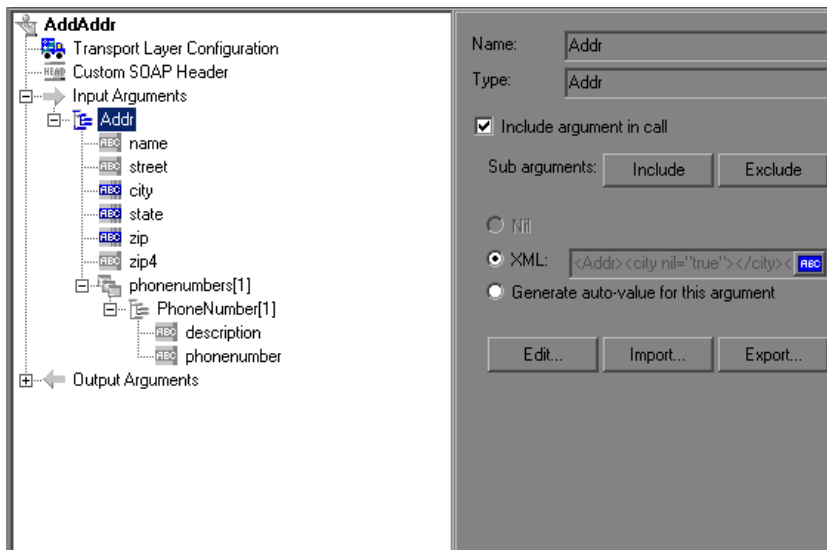
To include a specific optional argument in the service call, click the node and select **Include Argument in Call**. The nodes for all included arguments are colored in blue. Arguments that are not included are colored in gray.

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

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

To include a sub elements:

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

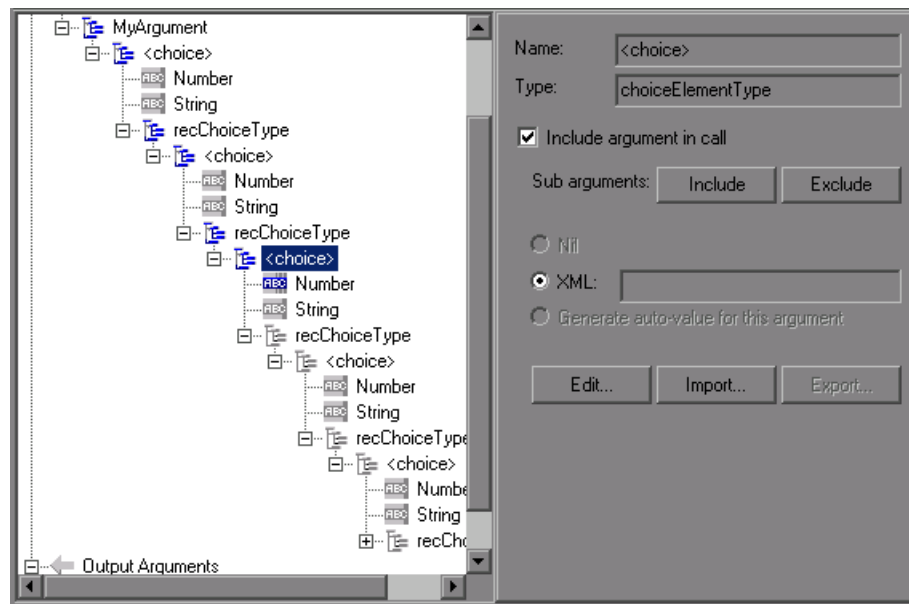


Recursive Elements

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

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

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



Choice Optional Elements

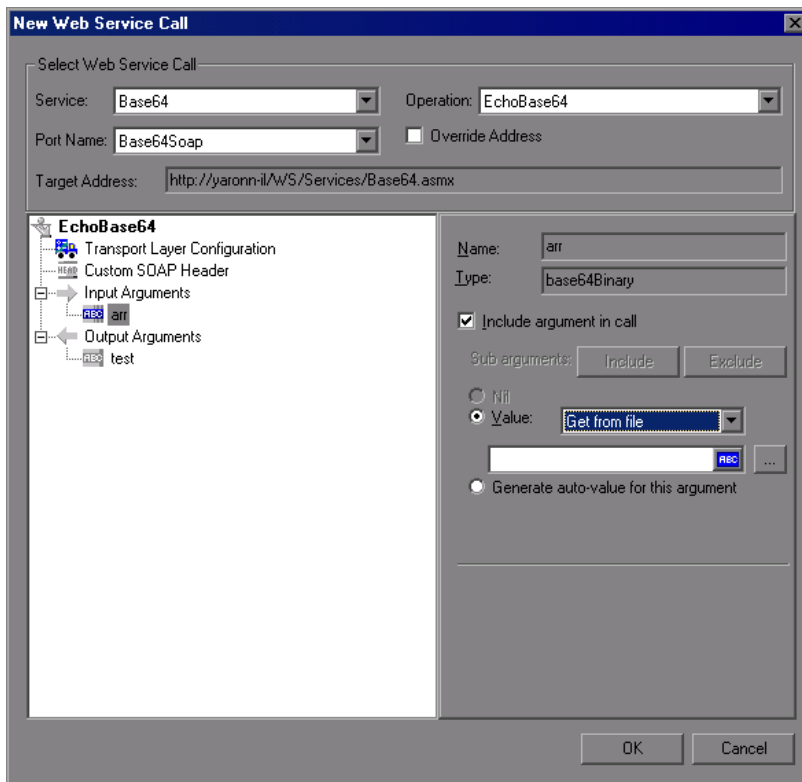
A Choice element in a WSDL defines a set of elements where only one of them appears in the SOAP message. In some cases, one of the Choice elements is optional, while the others are not. In Service Test, you can select the Choice element and still prevent its optional element from appearing in the SOAP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

Base 64 Encoding

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

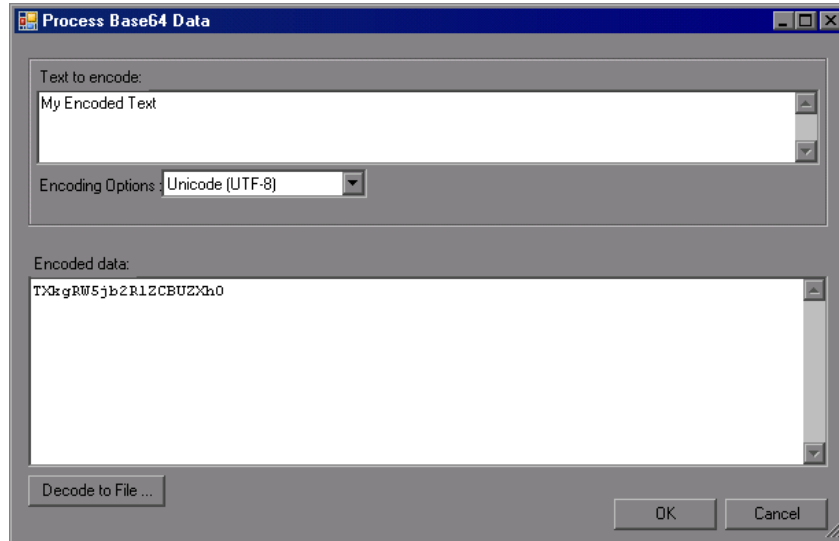
When VuGen detects a WSDL element of **base64Binary** type, it lets you provide an encoded value. You can specify a value in two ways:

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



To specify a **base64Binary** value:

- 1 Select the **Value** option.
- 2 To specify a file, select **Get from file** and locate the file using the Browse button below.
- 3 To specify text, select the **Embed encoded text** option and click the Browse button below. The Process Base64 Data dialog box opens.



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

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

Click **Encode**.

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

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

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

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

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

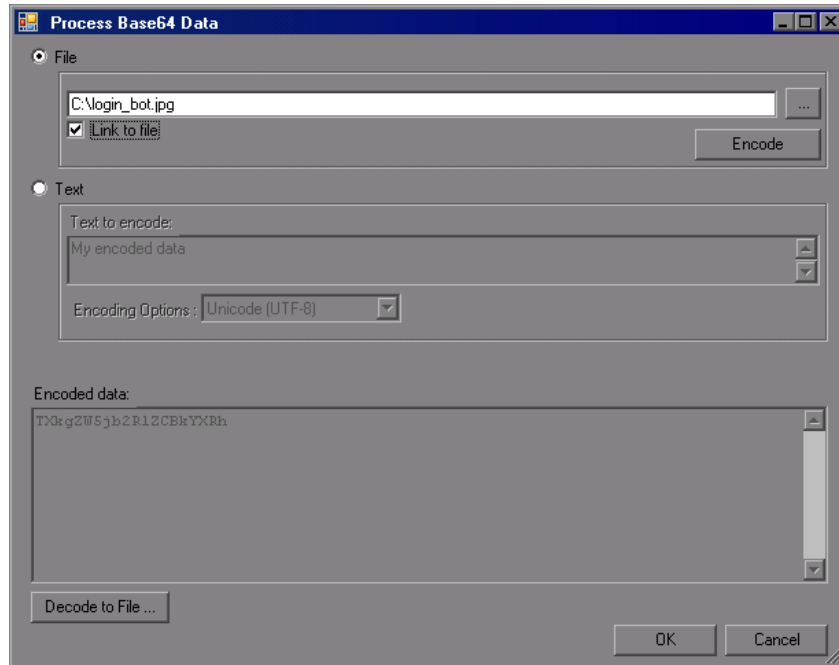
Setting a Parameter Value for Base64Binary Data

To set the Base64 argument value to a parameter, create a new parameter of File type, or XML type for Complex arguments. For more information, see "Setting Properties for XML Parameters" in *Volume I-Using VuGen*.

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

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

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



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

To open the Process Base64 dialog box:

- 1** Create a new parameter for a complex argument.
- 2** In the grid view (XML parameters or checkpoints), click the Browse button to the right of the value box. The Process Base64 dialog box opens.
- 3** Specify **File** or **Text** as a source for the value.

- 4 If you chose **File** as the Source, specify whether or not to link to a file.
- 5 To decode an encrypted value, for example, a value obtained during replay, click **Decode to File**. For more information, see below.

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

Decoding to a File

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

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

To validate images using decoding:

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

Attachments

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

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

The formats used for passing a SOAP message together with binary data are MIME (Multipurpose Internet Mail Extensions) and the newer, more efficient DIME (Direct Internet Message Encapsulation) specifications. 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 222.

VuGen supports the sending and receiving of attachments with SOAP messages. You can send Input (Request) or save Output (Response) attachments.

To add or save attachments, select an operation or a method in the left pane to which the attachments will be associated. You can add both input or output attachments

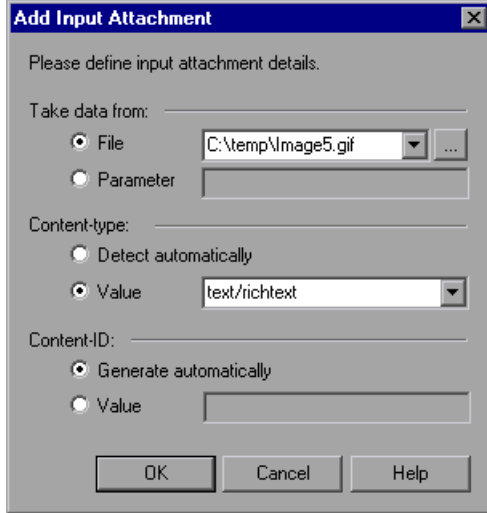
Input Attachments

Input attachments are added to the request message.

To add an attachment to the request:

- 1 Select the operation in the left pane, to which you want to add the attachment.
- 2 In the right pane, select **Add to request (Input)**. VuGen prompts you to enter information about the attachment and adds it to the method's tree structure.

The Add Input dialog box opens.



Specify the following information:

- ▶ **Take data from.** The location of the data. This can be a file or a parameter that contains the binary data.
 - ▶ **File.** You can specify the file location in two ways:
 - ▶ **Absolute Path:** The full path of the file. Note that this file must be accessible from all machines running the script.
 - ▶ **Relative Path:** (recommended) A file name. Using this method, during replay, VuGen searches for the attachment file in the script's folder. To add it to the script's folder, select **File > Add Files to Script** and specify the file name.
 - ▶ **Parameter.** You specify the name of a parameter containing the data.
- ▶ **Content-type.** The content type of the file containing the data. The **Detect Automatically** option instructs VuGen to automatically determine the content type. You can also select from a list of the common content types in the **Value** box, such as `text/html`, and `image/gif` or type in another content type.

- **Content-ID.** The ID of the content. By default, VuGen generates this automatically by VuGen and serves as a unique identifier for the attachment. Optionally, you can specify another ID in the **Value** box.

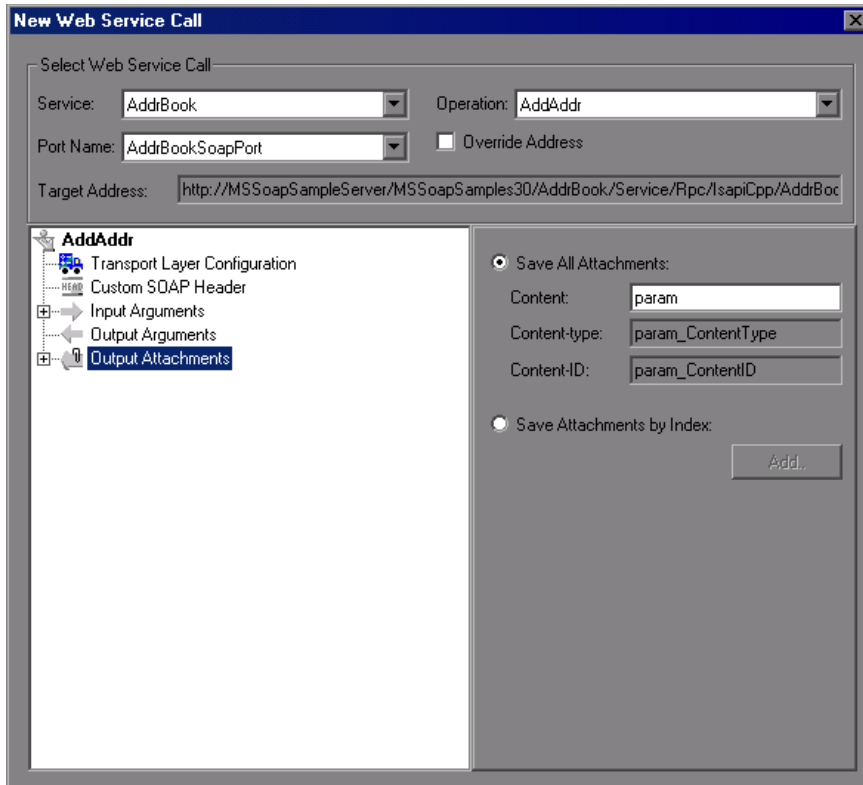
Output Attachments

Output attachments are added to the response message.

To save the response as an attachment:

- 1** Select the operation in the left pane, for which you want to save the response.
- 2** In the right pane, select **Save received (Output)**. VuGen adds an Output Attachment node to the method's tree structure in the left pane.

- 3 Select the desired option: **Save All Attachments** or **Save Attachment by Index** based on their index number—beginning with 1.



When you specify **Save All Attachments**, VuGen creates three parameters for each attachment based on the parameter name that 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.

To edit the properties of either an Input or Output attachment, click the attachment in the left pane, and enter the required information in the right pane.

SOAP Headers

This view is available when you select **SOAP header** in the method's tree view. The right pane lets you indicate whether or not to use SOAP headers. To use them, select **Use SOAP header**. Note that you must individually specify SOAP headers for each element. You can import XML code for the SOAP header, or compose your own using the Edit XML option. For more information, see "Editing an XML Tree" on page 118.

Working with the XML

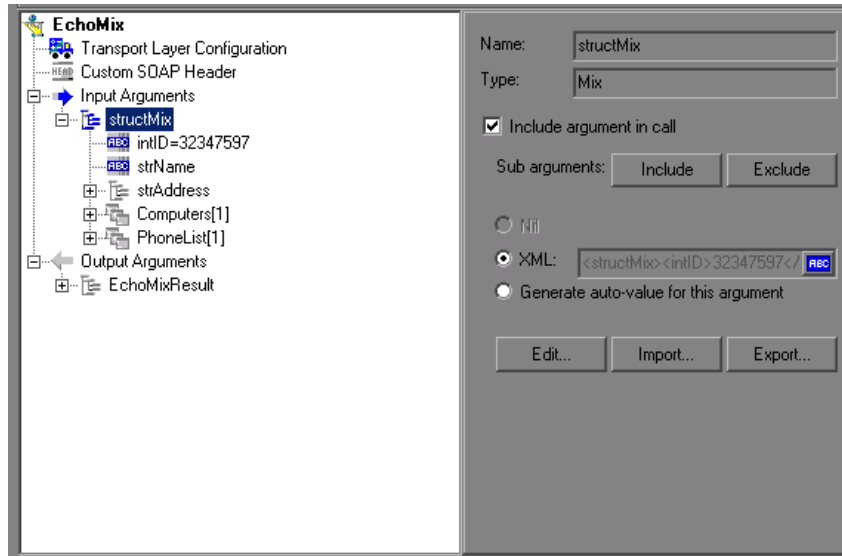
Web Services allow you to view and edit your XML.

The following sections describe:

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

Editing an XML Tree

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

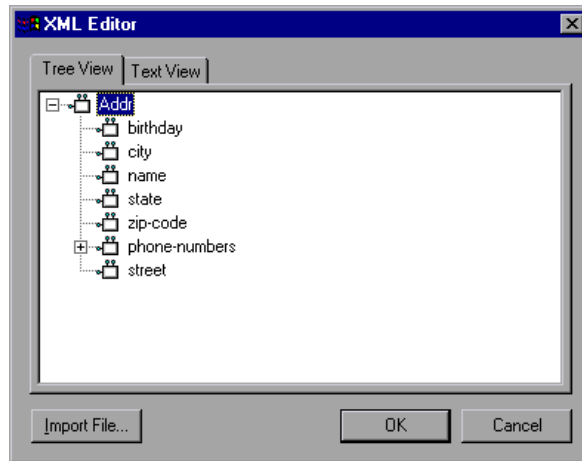


Entering the values for the XML elements is a tedious and error-prone task. VuGen provides you with an interface that simplifies the task of entering, saving, and restoring the information. Once you enter the data manually, you can save it to an XML file using the **Export** option. For subsequent tests,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

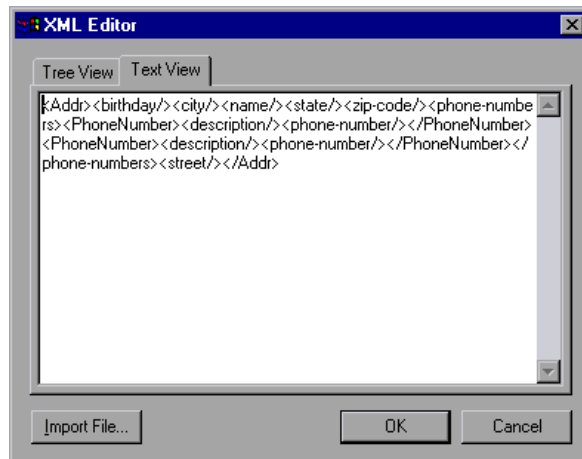
To edit XML strings:

- 1 Select the Web Service call whose element you want to modify and click the **Step Properties** tab.
- 2 In the method's tree hierarchy, click on a complex type or array argument. The right-most pane shows the XML code as a single string. Select the **XML** option.
- 3 To edit the XML code for that entry, click **Edit**. VuGen may issue a warning indicating that only changes to element values and the number of array elements will be saved—not changes in the XML elements themselves.

- 4 Click **OK**. The XML Editor dialog box opens.



- 5 In the XML Tree view, double-click on a node to open its property dialog box. Edit the value as required. Click **OK** to save the new values.
- 6 To edit the code in text mode, click the **Text View** tab. Edit the XML code manually. Click **OK** to save the changes.



- 7 To import a previously saved XML file, click **Import** and specify the file's location. Edit the file in the XML Editor dialog box.

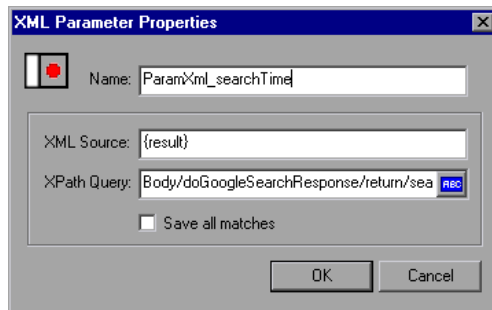
- 8 To save your XML data to a file so it can be used for other tests, click **Export** and specify a location.

Saving and Copying the SOAP Response

In addition to saving the input argument values as an XML type parameter, you can also save the SOAP response to a parameter or copy it for use within an editor.

To save the SOAP response to a parameter:

- 1 Switch to the **SOAP Snapshot** tab and select the parent or child element whose value you want to parameterize.
- 2 Select **Save XML in parameter** from the right-click menu. The XML Parameter Properties dialog box displays the properties of the selected XML element.



- 3 Specify a name for the XML parameter, and click **OK**.

To copy the XML structure for use within another editor, select **Copy XML** from the right-click menu.

7

Web Services - Preparing for Replay

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

This chapter includes:

- About Preparing Web Services Scripts for Replay on page 121
- Checkpoints on page 122
- Web Services JMS Run-Time Settings on page 123
- Using Web Service Output Parameters on page 126
- Handling Special Cases on page 130

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

About Preparing Web Services Scripts for Replay

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

You can enhance it with custom error and log messages or with transactions. See Chapter 6, "Enhancing Vuser Scripts" in *Volume I-Using VuGen* for more information.

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

You can configure run-time settings to help you emulate real users more accurately. These settings include general run-time settings (iteration, log, think time, and general information). Web Services specific settings relate to the JMS transport method, and are described in "Web Services JMS Run-Time Settings" on page 123.

For more information about the general run-time settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

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

Checkpoints

In functional testing, one of the most important tasks is to check the response from the server to confirm that your test performed the actions correctly. In Web Services, the response can contain several arguments, each containing several data items. The **Checkpoint** tab is a central point for defining the required response values for your test.

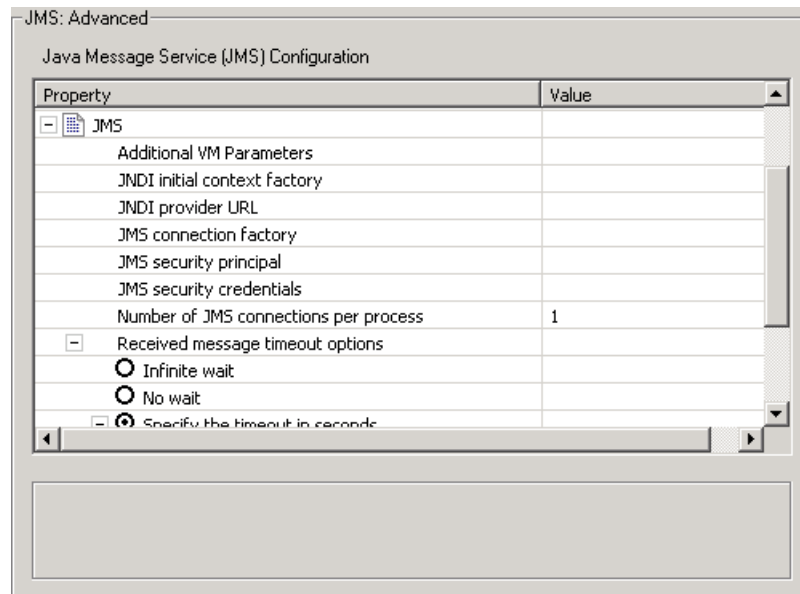
Checkpoint validation is only available with HP Service Test or HP LoadRunner with a Service Test license. For more information, contact HP support.

Web Services JMS Run-Time Settings

To use JMS as a transport for Web Service calls, there are several resources that need to be allocated and configured. Those resources include the JVM, JNDI initialization parameters, JMS resources, and timeout values. For information on setting the transport level, see Chapter 11, "Web Services - Transport Layers and Customizations."

VuGen lets you configure some of those resources through the run-time settings.

You can set options in the area of VM (Virtual Machine), the JMS connections, and message timeouts.



VM

- **Use external VM.** Enables you to select a VM (Virtual Machine) other than the standard one. If you disable this option, users use the JVM provided with VuGen.
- **JVM Home.** The location of the external JVM. This should point to the JDK home directory, defined by JDK_HOME. VuGen supports JDK 1.4 and above.

- ▶ **Classpath.** The vendor implementation of JMS classes together with any other required supporting classes, as determined by the JMS implementation vendor

JMS

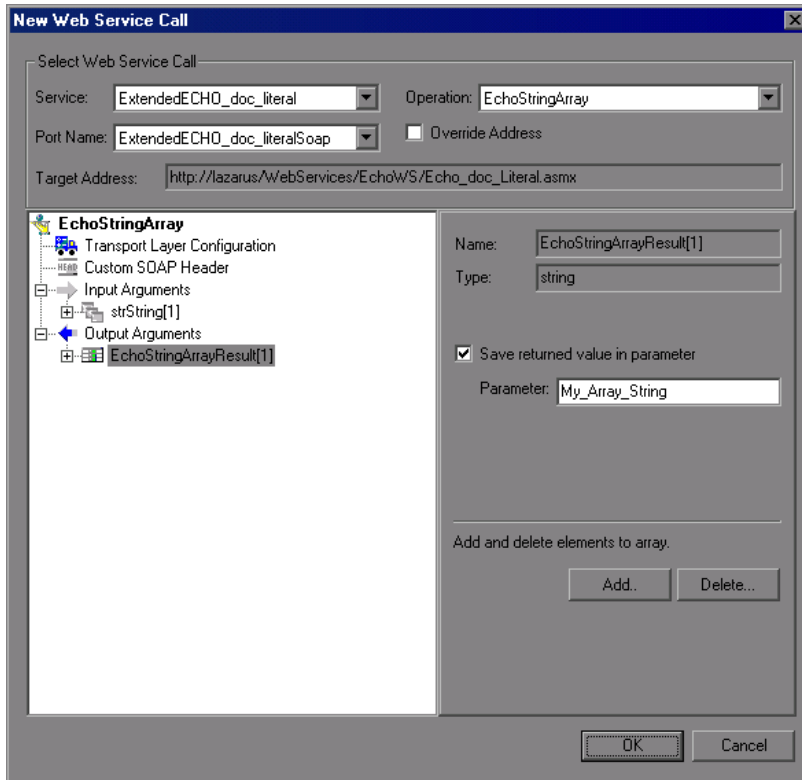
- ▶ **Additional VM Parameters.** Extra parameters to send to the JVM such as Xbootclasspath, and any parameters specified by the JVM documentation.
- ▶ **JNDI initial context factory.** The fully qualified class name of the factory class that will create an initial context. Select a context factory from the list or provide your own.
- ▶ **JNDI provider.** The URL string of the service provider. For example:
 - Weblogic - t3://myserver:myport
 - Websphere - iiop://myserver:myport
- ▶ **JMS connection factory.** The JNDI name of the JMS connection factory. You can only specify one connection factory per script.
- ▶ **JMS security principal.** Identity of the principal (for example the user) for the authentication scheme.
- ▶ **JMS security credentials.** The principal's credentials for the authentication scheme.
- ▶ **Number of JMS connections per process.** The number of JMS connections per `mdrv` process, or `Vuser`. All `Vusers` sharing a connection will receive the same messages. The default is 1, and the maximum is 50 `Vusers`. The fewer connections you have per process, the better your performance.

- ▶ **Receive message timeout options.** The timeout for received messages. The default is **No wait**.
 - ▶ **Indefinite wait.** Wait as long as required for the message before continuing.
 - ▶ **No wait.** Do not wait for the Receive message, and return control to the script immediately. If there was no message in the queue, the operation fails.
 - ▶ **Specify the timeout in seconds.** Manually specify a timeout value for the message. If the timeout expired and no message has arrived, the operation fails. (default)
 - ▶ **User defined timeout.** Specify the amount of seconds to wait for the message before timing out. The default is twenty seconds.
- ▶ **Automatically generate selector.** Generates a selector for the response message with the correlation ID of the request (**No** by default). Each JMS message sent to the server has a specific ID. Enable this option if you want VuGen to automatically create a selector that includes the message ID.

Using Web Service Output Parameters

In certain cases, you may need to use the result of one Web Service call as input for another. To do this, you save the result to an output parameter and reference it at the required point.

In the following example, the output argument is saved to a parameter, **My_Array_String**.

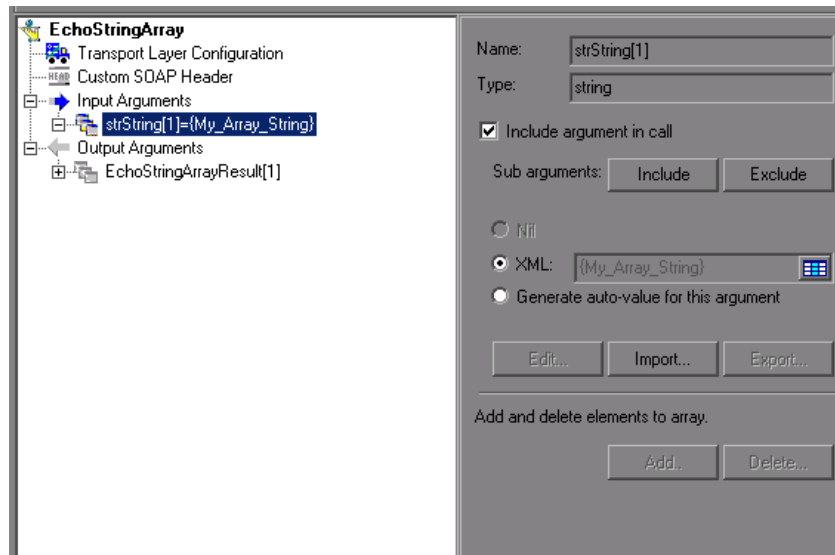


The script shows the saved output parameter as a result argument:

```
web_service_call( "StepName=EchoStringArray_101",
"SOAPMethod=ExtendedECHO_doc_literal.ExtendedECHO_doc_literalSoap.EchoStringArray",
    "ResponseParam=response",
    "Service=ExtendedECHO_doc_literal",
    "Snapshot=t1169994766.inf",
    BEGIN_ARGUMENTS,
    "xml:strString=<strString><string></string></strString>",
    END_ARGUMENTS,
    BEGIN_RESULT,
    "EchoStringArrayResult*[1]=My_Array_String",
    END_RESULT,
    LAST);
```

For information on saving results to parameters, see "Saving Output Parameters" on page 128.

After you save an output parameter, it becomes available for parameter substitution or for other referencing, such as evaluating it and printing its value. In the following example, the saved output parameter, **My_Array_String**, is used as an input argument for a subsequent Web Service call.



For information on using saved output parameters, see "Using Saved Parameters for Input" on page 129.

Saving Output Parameters

You can save multiple result arguments to parameters through the Properties dialog box, or by manually editing them in the script code.

You can also save result parameters from XML actions, and use them as input arguments. For example, if you save the result parameter for **lr_xml_insert**, you can reference the saved parameter in a subsequent Web Service call. For more information, see the *Online Function Reference*.

To save an output parameter:

1 View the script in Tree view.

Make sure you are in Tree view. Otherwise, select **View > Tree view**.

2 Check for a Service.

Click the **Manage Services** button to verify that you have imported at least one service. To import a new service, click **Import** in the Service Management dialog box.

3 View the step's properties.

For a new Web Service call, click **Add Service Call**.

For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

4 Select the output argument.

In the left pane, select the output argument whose value you want to save to a parameter.

5 Enable the saving of the output parameter.

In the right pane, select **Save returned value in parameter**. Accept the default name or specify a custom name.

Using Saved Parameters for Input

After saving output parameters, you can use them in subsequent Web Service calls.

You can also use saved result parameters from XML actions as input. For example, if you saved the result parameter for `lr_xml_insert`, you can reference it in a subsequent Web Service call. For more information, see the *Online Function Reference*.

To use a saved parameter for input:

1 View the step properties in Tree view.

For a new Web Service call, click **Add Service Call**.

For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

2 Select the input argument.

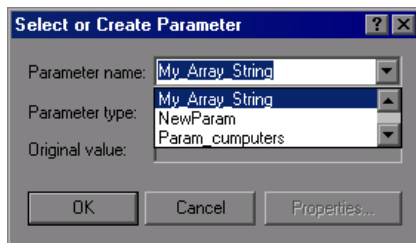
In the left pane, select the input argument whose value you want to replace with a previously saved output parameter.

3 Open the Select Parameter dialog box.

In the right pane, select **Value**, and click on the ABC icon adjacent to the **Value** box. The Select or Create Parameter box opens.

4 Select an output parameter.

Select the desired output parameter from the drop-down list and click **OK**.



To specify an input parameter in Script view, select the value you want to replace and select **Use Existing Parameters** from the right-click menu. Select one of the available parameters.

Note: If you modify an output parameter name in Script view, it will not be updated in the parameter list until you switch to Tree view.

Handling Special Cases

This section provides guidelines for running scripts in special cases.

Any Type XSD

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

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

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

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

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

8

Web Services - Database Integration

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

This chapter includes:

- ▶ About Database Integration on page 131
- ▶ Connecting to a Database on page 132
- ▶ Using Data Retrieved from SQL Queries on page 135
- ▶ Validating Database Values after a Web Service Call on page 138
- ▶ Checking Returned Values Through a Database on page 140
- ▶ Performing Actions on Datasets on page 142

About Database Integration

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

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

The database integration is useful in the following scenarios:

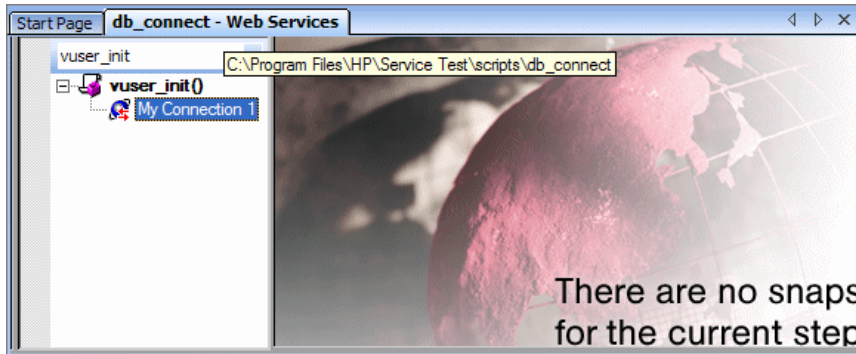
- ▶ Using Data Retrieved from SQL Queries
- ▶ Validating Database Values after a Web Service Call
- ▶ Checking Returned Values Through a Database

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

Connecting to a Database

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

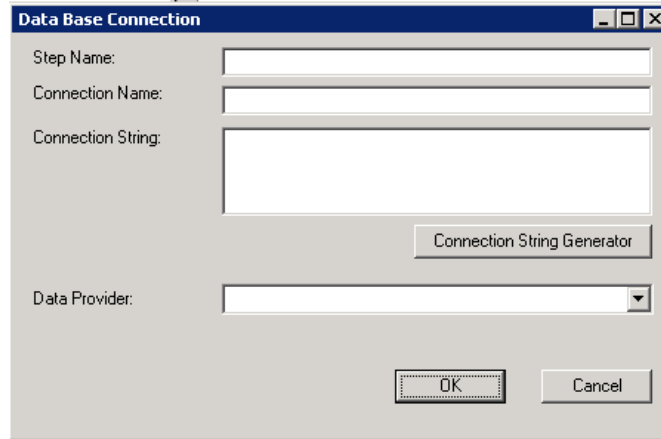
When running your script with iterations, virtual users only repeat the **Action** section of the script. If you include the database connection step in the **Action** section, the test will repeat it for each iteration. Virtual Users only repeat the **Action** section of the script, but not the **vuser_init**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



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

To add a database connection step through Tree view:

- 1** Select **View > Tree View** to enter Tree view (if it is not already visible).
- 2** Select the desired section: **vuser_init** or **Action**. We recommend placing the connection step in the **vuser_init** section. For more information, see below.
- 3** Select **Insert > New Step**. Choose the **Database: Connect** step. The Database Connection dialog box opens.

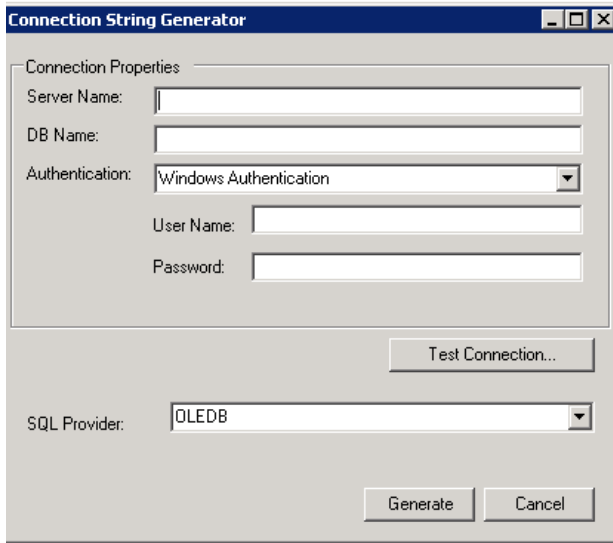


The screenshot shows a dialog box titled "Data Base Connection". It has a standard Windows-style title bar with minimize, maximize, and close buttons. The dialog contains the following elements:

- Step Name:** A single-line text input field.
- Connection Name:** A single-line text input field.
- Connection String:** A multi-line text area for entering the connection string.
- Connection String Generator:** A button located to the right of the Connection String text area.
- Data Provider:** A dropdown menu.
- OK and Cancel:** Two buttons at the bottom of the dialog.

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

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



The screenshot shows a dialog box titled "Connection String Generator". It has a "Connection Properties" section with the following fields: "Server Name:" (text box), "DB Name:" (text box), "Authentication:" (dropdown menu showing "Windows Authentication"), "User Name:" (text box), and "Password:" (text box). Below these fields is a "Test Connection..." button. At the bottom of the dialog is an "SQL Provider:" dropdown menu showing "OLEDB", and two buttons: "Generate" and "Cancel".

- 6 Indicate the connection properties:
 - **Server Name**
 - **Database Name**
 - **Authentication** method: Windows Authentication or User/password.
 - **Username** and **Password**
- 7 Click **Test Connection** to verify that the information you provided is correct.
- 8 Select an **SQL Provider**, OLEDB or SQL, and click **Generate**.

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

Using Data Retrieved from SQL Queries

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

The following table shows a typical flow of the script:

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

You can iterate through the results in two ways:

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

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

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

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

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

  return 0;
}
```

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

```
vuser_end()
{

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

  return 0;
}
```


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

```

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

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

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

    return 0;
}

```

Validating Database Values after a Web Service Call

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

The following table shows a typical flow of the script:

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

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

The following example illustrates this process of checking the data:

```

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

/* Query the database by the 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**.

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**).

9

Web Services - Security

Advanced users can customize Web Service calls by setting the transport layer properties and security policies, and by writing user handlers to define the behavior of the Web Service calls.

This chapter includes:

- About Adding Security for Web Service Testing on page 143
- Adding Security to a Web Service Call - a General Workflow on page 145
- Security Tokens and Encryption on page 147
- Setting SAML Options on page 152
- Examples Using `web_service_set_security` on page 155
- Customizing Your Security on page 159

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

About Adding Security for Web Service Testing

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

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

For more information on tokens, see below. For more information on SAML, see "Setting SAML Options" on page 152.

To learn more about customizing WS-Security and testing with WCF, see Chapter 10, "Web Services - Advanced Security and WS Specifications."

Note: If your WSDL is located in a secure location, you must provide the security information through the Manage Services dialog box. For more information, see "Specifying WSDL Connection Settings" on page 50.

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Legacy security model using `web_service_set_security`. For information on the Scenario model, see Chapter 10, "Web Services - Advanced Security and WS Specifications."

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

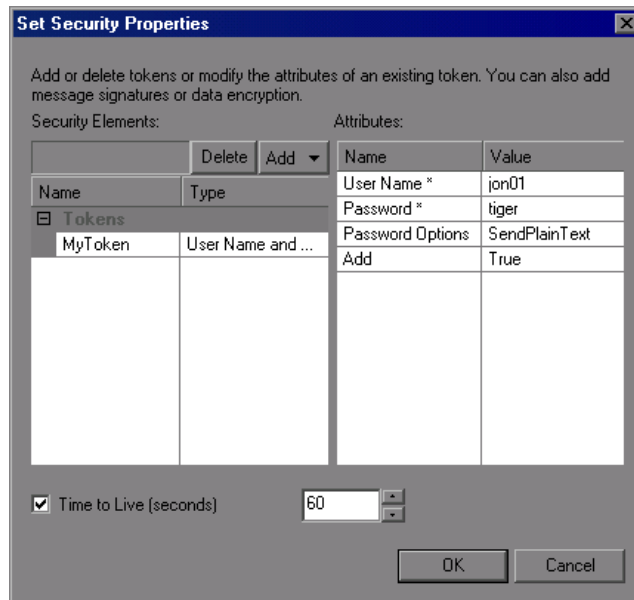
Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework, such as Axis2 or Metro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

Adding Security to a Web Service Call - a General Workflow

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

To add Web Service security:

- 1** Place the cursor at the point at which you want to add the security settings. In most cases, we recommend that you place it in the `vuser_init` section so that the security scope will be applied to the whole script. If you only want the security for specific calls, place it at the desired location.
- 2** Select **Insert > New Step** to open the Add Step dialog box.
- 3** Select **Web Services Set Security** and click **OK**. The Set Security Properties box opens.



- 4 Click **Add** to add a new token. The Add Token dialog box opens.

Select a token type and give it an arbitrary name. VuGen uses this as an ID for the newly defined token. Note that an asterisk denotes a mandatory field.

Type: * User Name and Password

Logical Name: *

Properties:

User Name: *

Password: *

Password Options:

Add: True

OK Cancel Help

- 5 Select a token type. Common tokens are Username and X.509 certificate. For information about the token types, see "Security Tokens and Encryption" on page 147.

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

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

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

- 6 To specify a time for which the message packet is considered valid, select **Time To Live** and specify the time in seconds.
- 7 Click **Add** to add a message signature and encryption if they are required. Both signatures and encryptions require you to specify a token previously defined as the encrypting/signing token. See the "Examples Using web_service_set_security" on page 155 to learn how to encrypt or sign a specific XPath in the SOAP.

- 8 To cancel the security settings at a specific point within the script, add a **Web Service Cancel Security** step at the desired point.
- 9 Click **OK**. VuGen inserts a Web Services Set Security step at the location of the cursor.

"Examples Using web_service_set_security" on page 155 illustrates some of the common security settings.

Security Tokens and Encryption

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

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

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

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

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

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

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

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

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

When you add an X.509 token to the Vuser script, you specify the **Logical Name**, **Store Name**, **Key identifier type**, **Key identifier value**, and **Store Location** arguments.

- ▶ **Kerberos Ticket/Kerberos2 Ticket.** (for Windows 2003 or XP SP1 and later) The Kerberos protocol is used to mutually authenticate users and services on an open and unsecured network. Using shared secret keys, it encrypts and signs user credentials. A third party, known as a KDC (Kerberos Key Distribution Center), authenticates the credentials. After authentication, the user may request a service ticket to access one or more services on the network. The ticket includes the encrypted, authenticated identity of the user. The tickets are obtained using the current user's credentials.

VuGen supports tokens based on both Kerberos and Kerberos2 security tokens. The primary difference between the 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 more information about configuring tokens, see the *Online Function Reference* (**Help > Function Reference**).

Adding the Security Policy

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

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

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

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

Working with Message Signatures and Encrypted Data

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

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

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

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

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

VuGen allows you to supply information about the encryption and message signatures.



Note that parameterization is not supported for message signatures and encryption arguments. For more information on adding message signatures and encryption to your script, see below.

Setting SAML Options

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

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

You can set the SAML settings for an entire script or part of the script. To set SAML security, add a **Web Services Set Security SAML** step. To remove the security, insert a **Web Services Cancel Security SAML** step.

Note: You cannot apply SAML security and the standard Web Service (a **Web Service Set Security** step) security to the same step. To cancel Web Service security, insert a **Web Service Cancel Security** step.

Signing an SAML Assertion

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

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

Policy Files

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

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

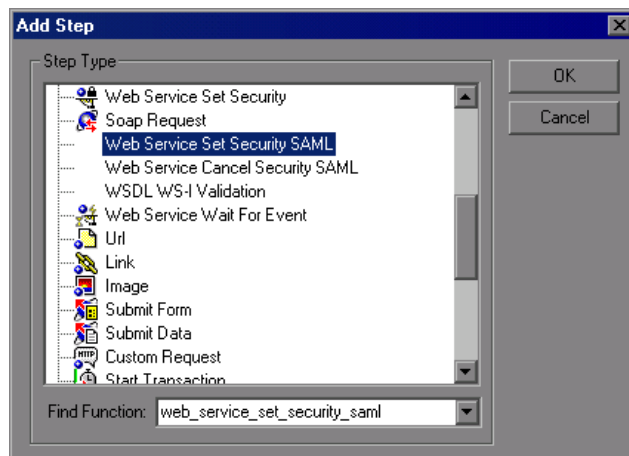
You can modify the policy file to include values for the security parameters, such as username and certificate information. When adding a SAML security step to your script, if you explicitly specify values for the security arguments, they override the values in the policy file.

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

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

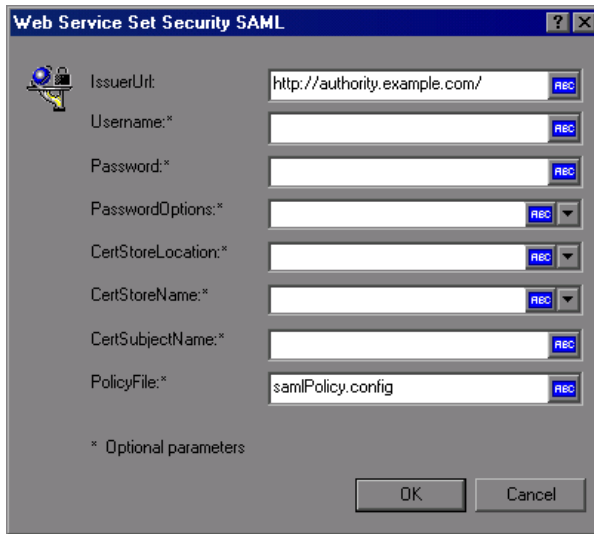
To add SAML security:

- 1 Click at the appropriate location in your script. To apply the security to the entire script, place the cursor at the beginning of the script.
- 2 Select **Insert > New Step** to open the Add Step dialog box.



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

Enter the desired information. If you enter values into this dialog box, they override any values in the policy file. You must provide an Issuer URL, also known as the **STS URL**.



To use a different policy file, specify it in the **Policy File** box. Specify a full path, or a file location relative to the script's path.

4 To remove the security, select **Web Services Cancel Security SAML**. The security is cancelled from that point onward.

For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference**, or click **F1** on the function).

Examples Using `web_service_set_security`

This section illustrates several common security scenarios.

Authenticating with a Username Token

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

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

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

Encrypting with a Certificate

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

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

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

Since we want to generate a Subject Key Identifier, we set the Add value to **False**. For more information, see "Using SubjectKeyIdentifier" on page 159.

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=pO10+1iuotKLIO91nhjDg5reEw0=",  
  "StoreLocation=CurrentUser", "Add=False",  
  ENCRYPTED_DATA, "UseToken=serviceCert",  
  LAST);
```

Customizing Your Security

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

Using SubjectKeyIdentifier

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

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

Alternatively, you can configure this setting in the script:

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

If you are using a SKI (Subject Key Identifier) you may also need to modify the **useRFC3280** settings as described in "Customizing WS-Security" on page 162.

Username Customization

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

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

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

Name	Meaning	Possible values
IsNonceIncluded	Should the username token contain a nonce	True (default) or False
TimestampFormat	Should the username token contain a timestamp. If so, in what format	<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

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

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


Customizing Encryption

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

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

Name	Meaning	Possible values
EncryptionType	What to encrypt	<ul style="list-style-type: none"> ▶ Element ▶ Content (default)

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

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

Customizing WS-Security

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

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

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

Set the element values as required:

Name	Meaning	Possible values
verifyTrusy	Whether to check sent/received x.509 certificate's validity	<ul style="list-style-type: none"> ▶ True (default) ▶ False
sessionKeyAlgorithm	The algorithm that the session symmetric key will use to encrypt the message	<ul style="list-style-type: none"> ▶ AES128 ▶ AES192 ▶ AES256 ▶ TripleDES
keyAlgorithm	The algorithm used by the public key to encrypt the session key	<ul style="list-style-type: none"> ▶ RSA15 ▶ RSAOAEP
useRFC3280	Whether to generate subject key identifiers that are interoperable and not windows specific	<ul style="list-style-type: none"> ▶ True ▶ False (default)

10

Web Services - Advanced Security and WS Specifications

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

This chapter includes:

- About Advanced Security and WS Specifications on page 164
- Choosing a Security Model on page 165
- Setting the Security Scenario on page 166
- Scenario Types on page 170
- Specifying Scenario Information on page 172
- Selecting a Certificate on page 177
- Advanced Settings on page 179
- Customizing Your Security Model on page 184
- Simulating Users with Iterations on page 188
- Tips and Guidelines on page 190

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

About Advanced Security and WS Specifications

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

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

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

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

For a "How To" guide on selecting a scenario, see "Tips and Guidelines" on page 190.

Choosing a Security Model

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Scenario security model. The Legacy model refers to the manual addition of **Web Service Set Security** steps, or the `web_service_set_security` function. For information on the Legacy model, see Chapter 9, "Web Services - Security."

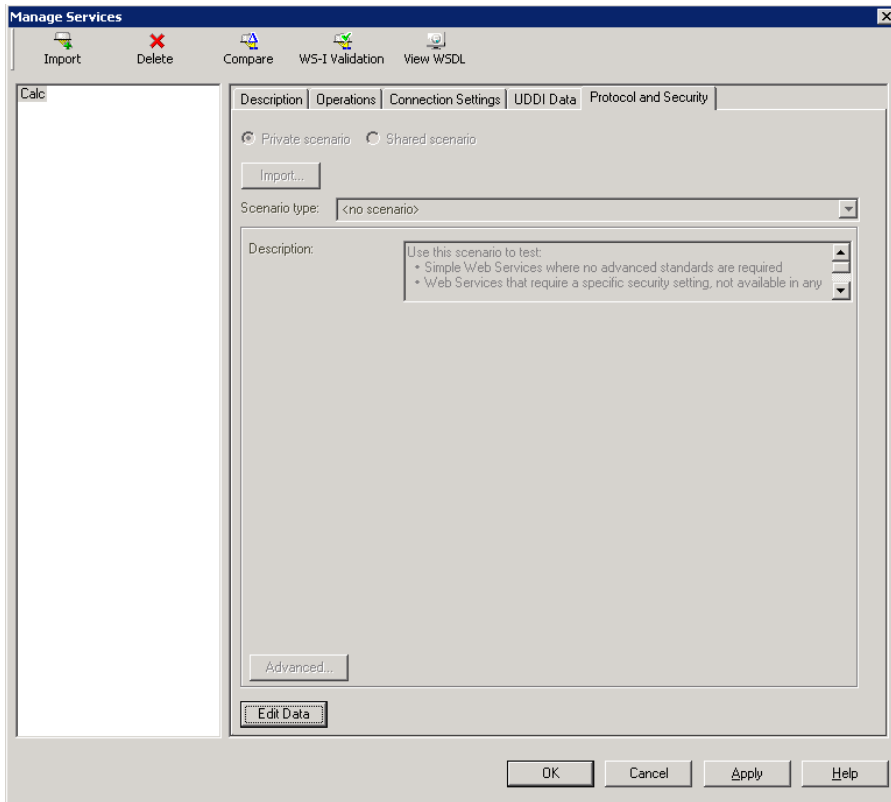
The following table lists the considerations for using each of the models.

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

Setting the Security Scenario

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

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



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

Setting the Security Scenario for a Service

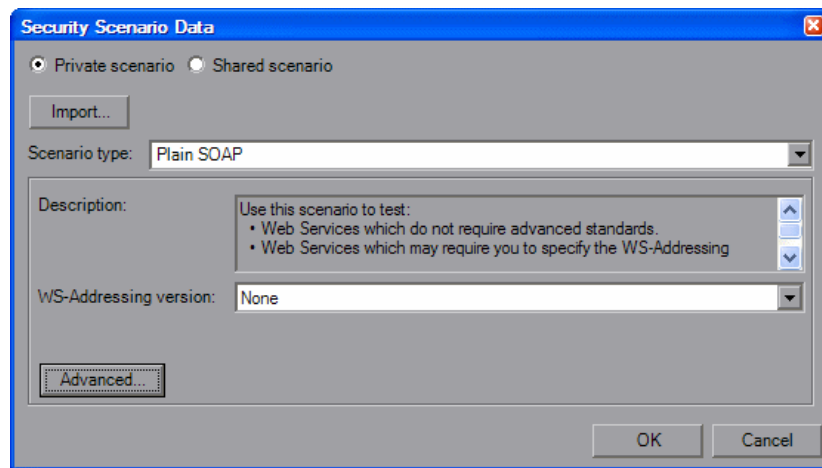
To assign a security scenario to a specific service, use the Manage Services window. The **Protocol and Security** tab contains the interface to create and view security scenarios for individual services.

You can select a scenario in three ways:

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

To create a security scenario for a specific service:

- 1** Click **Manage Services**. In the left pane, select the service for which you want to set the security scenario. If necessary, import a service, as described in "Importing Services" on page 45.
- 2** Select the **Protocol and Security** tab and click the **Edit Data** button. The Security Scenario Data dialog box opens.



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

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

In the Scenario type box, choose a scenario. The scenario types are described in "Scenario Types" on page 170.

Specify the required values for your scenario. For details, see "Specifying Scenario Information" on page 172.

b To use an existing scenario with the ability to modify it, choose **Private scenario**. Click **Import**. In the Shared Scenario dialog box, select a stored scenario. If required, modify the settings as described in "Specifying Scenario Information" on page 172.

c To use an existing scenario without the option of changing it, choose **Shared Scenario**. Use the Browse button to open the Shared Scenario dialog box and select a stored scenario.



Click **OK**. Your service is now loaded with a security scenario. If someone modifies the scenario file at its source, it will affect your script.

4 Click **Advanced** to configure the Proxy, Encoding, and other advanced setting (optional). For most scenarios, the default settings are ideal. For information about these settings, see "Advanced Settings" on page 179.

5 Click **OK** to close the dialog box and save your script.

Saving Security Scenarios for Collaboration

One user can customize a security scenario and make it available to others. Using the Security Scenario Editor, you customize your settings and save them to a scenario file.

To create a new security scenario:

- 1** Choose **SOA Tools > Security Scenario Editor**.
- 2** Select a **Scenario type** and enter the relevant information.
- 3** When enabled, click **Advanced** to configure the Proxy, Encoding, and other setting as described in "Advanced Settings" on page 179. Click **OK** to close the dialog box.
- 4** For a new scenario, click **Save as**, and specify a file name and path for the scenario file.

To edit an existing stored scenario:

- 1** Choose **SOA Tools > Security Scenario Editor**.
- 2** Click the **Open** button and browse for an existing scenario file.
- 3** Modify the scenario settings as required.
- 4** Click the **Save** or **Save as** button.
- 5** When enabled, click **Advanced** to configure the Proxy, Encoding, and other advanced settings. For more information, see "Advanced Settings" on page 179.
- 6** Click **OK** to close the Security Scenario editor.

Scenario Types

The scenario describes the configuration of your Web Service. It contains information such as encoding, identities, proxy, and so forth. Service Test provides a Security Scenario editor that allows you to configure the settings for each scenario.

To determine the scenario that best fits your service, refer to the table below. If you are unsure which scenario to choose, we recommend that you use the **Custom Binding** scenario which allows you to test any service. For more information, see "Custom Binding" on page 176.

The following table lists the built-in scenarios and describes when to use them.

Scenario Name	When to use
<no scenario>	<ul style="list-style-type: none"> ▶ Default setting: if you have no need for a special security/protocols configuration, leave this value as is. Simple Web Services where no advanced standards are required. ▶ Scripts that use the legacy security model described in Chapter 9, "Web Services - Security." ▶ Web Services that require a specific security setting, not available in any of the existing scenarios ▶ If you select a built-in scenario and experience problems in replay, it is possible that no scenario is required and the problem is elsewhere. Reset the value to <no scenario>.
Plain SOAP	<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
MTOM	<ul style="list-style-type: none"> ▶ MTOM enabled Web services ▶ Web Services which may require you to specify the WS-Addressing version

The following table shows the scenarios for Web Services that utilize WCF. The WSHttpBinding-based scenarios are divided according to the way the client authenticates itself to the server. For example, if your client presents a user name and a password to the server, choose the **Username (message protection)** scenario. The user interface lets you provide the identity information in the form of a user name or a certificate as required.

WCF Scenario Name	When to use
WSHttpBinding - No Authentication	<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

Specifying Scenario Information

This section describes the values required for each of the security scenarios.

<no scenario>

Since you are not using a secure scenario, you do not have to specify any values.

Plain SOAP

For this type of scenario, if your service uses WS-Addressing, specify the version.

MTOM

For MTOM type scenarios, if your service uses WS-Addressing, specify the version.

No Authentication

In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication.

Specify only one of the following settings:

- ▶ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.

- **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 177. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information.

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Windows Authentication

This WCF scenario uses Windows Authentication.

You declare the expected identity of the server in terms of its **SPN** or **UPN** identities. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.

Certificate Authentication

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 177. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username Authentication (Message Protection)

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 177. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username (Transport Protection) Authentication

This WCF WSHttpBinding scenario enables SSL and authenticates the client with a user name and password on the message level.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

Federation

In the WSFederationHttpBinding scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server.

Therefore, two bindings are needed, one against the STS and another against the application server.

First, use the Security Scenario editor to define an STS binding. For more information, see "Saving Security Scenarios for Collaboration" on page 169. When setting the binding against the application server, specify this file in the **Referenced file** box.

For the Federation scenario, specify the following server information:

- **Transport.** HTTP or HTTPS
- **Encoding.** Text or MTOM

For the Federation scenario, specify the following security information:

- **Authentication mode.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, IssuedTokenOverTransport, or SecureConversation
- **Bootstrap policy.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, or IssuedTokenOverTransport

For the Federation scenario, specify the following identity information:

- **Server certificate.** Browse for a server certificate. For more information, see "Selecting a Certificate" on page 177.
- **Expected server DNS.** the expected identity of the server in terms of its DNS. This can be **localhost** or an IP address or server name.

For the Federation scenario, specify the following STS (Security Token Service) information:

- **Issuer address.** The address of the issuer of the STS. This can be **localhost**, an IP address, or a server name.

- ▶ **Referenced file.** The file that references the binding that contacts the STS (Security Token Service)

Custom Binding

The **Custom Binding** scenario enables the highest degree of customization. Since it is based upon WCF **customBinding**, it allows you to test most WCF services, along with services on other platforms such as Java that use WS - *<spec_name>* specifications.

Use the **Custom Binding** scenario to configure a custom scenario that does not comply with any of the predefined security scenarios.

For the Custom Binding scenario, specify the following server information:

- ▶ **Transport.** HTTP, HTTPS, TCP, or NamedPipe
- ▶ **Encoding.** Text, MTOM, or WCF Binary

Specify the following security information:

- ▶ **Authentication mode.** None, AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SecureConversation, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- ▶ **Bootstrap policy.** For SecureConversation type authentication, specify a bootstrap policy: AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- ▶ **Net security.** the network security. Select None, Windows stream security, or SSL stream security. For services with HTTP transport, leave the default value, **None**. To enable SSL for HTTP, choose the HTTPS transport.

If your Web Service uses Reliable messaging, enable the option, and select **Ordered** or **Not Ordered**.

Identities

Your security settings may require you to provide identity details for either the client and server, or both of them.

An example of identity details for the client, are user name/password or an **X.509** certificate.

For identity information, provide one or more authentication details as required by the service:

Username, Password, Server certificate or **Client certificate**. For information about choosing a certificate, see "Selecting a Certificate" on page 177.

Some scenarios require you to declare the expected identity of the server in terms of its DNS, SPN, or UPN identity.

- ▶ **DNS**. Provide the name of a server or use localhost.
- ▶ **SPN**. Provide the SPN identity in the domain\machine format.
- ▶ **UPN**. Provide the UPN identity in the user@domain format.

After setting the basic values, you can set advanced attributes as described in "Advanced Settings" on page 179.

Selecting a Certificate

This section describes how to select a certificate.

You can select a certificate from either a file or a Windows store.

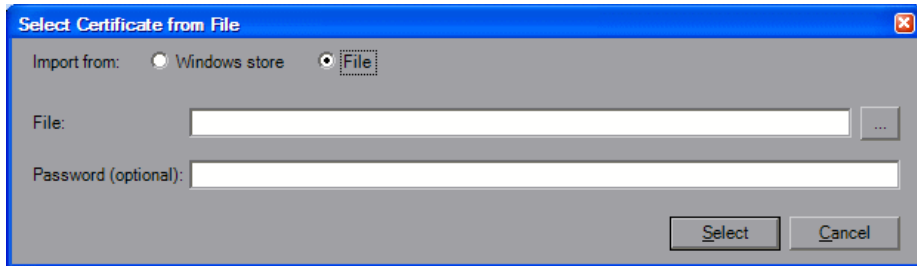
Certificates Stored in Files

In this case, the certificate is stored in a file.

To select a certificate from a file:

- 1** Click the Browse button adjacent to the Client Certificate or Server certificate box.

- 2 Choose **File**. Click the Browse button to the right of the File box and locate the certificate file.



- 3 If required, specify a password for the private key.
- 4 Click **Select**. The application now references the certificate file.

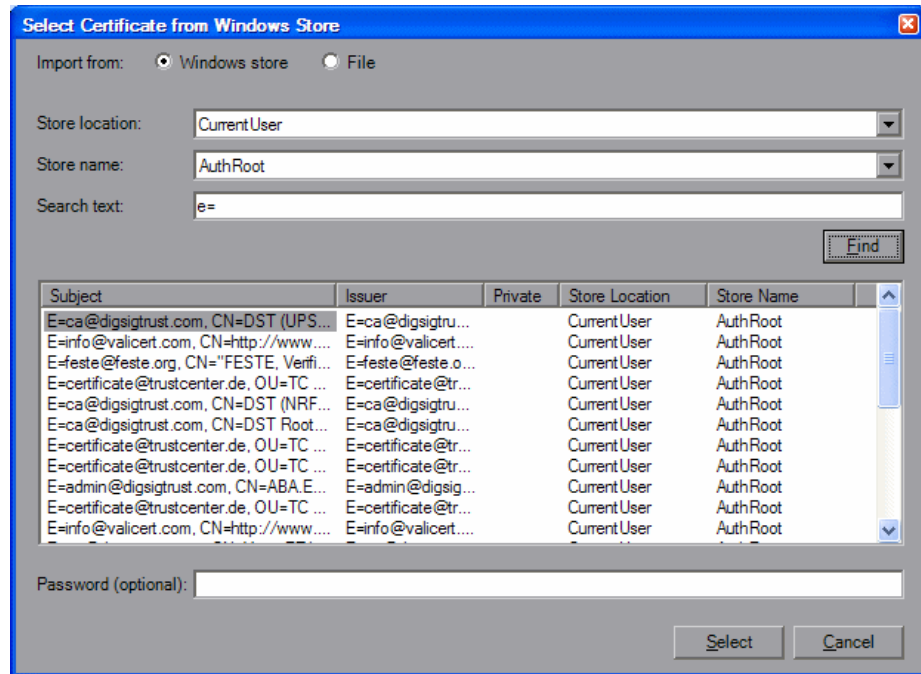
Certificates From a Windows Store

In this case, the location of the certificate is in a Windows store. The dialog box provides you with a mechanism to search for the certificate.

To select a certificate from a Windows Store:

- 1 Click the Browse button adjacent to the Client Certificate or Server Certificate box.
- 2 Choose **Windows Store**. Click the Browse button to the right of the File box and locate the certificate file.
- 3 Select a Store location: **All**, **CurrentUser**, or **LocalMachine** (optional).
- 4 Select a **Store name** from the drop-down list (optional).
- 5 To search for all certificates, leave the **Search text** box empty. To search for a specific certificate, specify a substring of the certificate name.

6 Click **Find** to generate the list of certificates found in the store.



7 If required, specify a password for the private key.

8 Select the certificate in the list and click **Select**. The application now references the chosen certificate.

Advanced Settings

This section describes the Advanced scenario settings. Using these settings, you can customize a security scenario in the following areas: Encoding, Advanced Standards, Security, or HTTP and Proxy.

Note that not all settings are relevant for all scenarios, so some of them might be disabled or hidden depending on the scenario.

Encoding

The Encoding tab lets you indicate the type of encoding to use for the messages: **Text**, **MTOM**, or **Binary**. The default is **Text** encoding.

For each of these encoding methods, you can choose a version of WS-Addressing:

- None
- WSA 1.0
- WSA 04/08

Advanced Standards

This tab lets you configure advanced WS- standards, such as Reliable Messaging.

If your service implements the **WS-ReliableMessaging** specification, enable the **Reliable Messaging** option and set the following options:

- **Reliable messaging ordered.** indicates whether the reliable session should be ordered
- **Reliable messaging version.** WSReliableMessagingFebruary2005 or WSReliableMessaging11

Security

The Advanced security settings correspond to the **WS-Security** specifications.

For security scenarios that are based upon WCF WSHttpBinding, you can indicate the following settings:

- **Enable secure session.** Establish a security context using the WS-SecureConversation standard.
- **Negotiate service credentials.** Allow WCF proprietary negotiations to negotiate the service's security.

For **WSHttpBinding**, **Custom Binding**, or **WSFederationHttpBinding** WCF type scenarios, you can set the default algorithm suite and protection level:

Attribute	Meaning	Possible Values
Default Algorithm Suite	the algorithm to use for symmetric/asymmetric encryption. These are the values from the SecurityAlgorithmSuite configuration in WCF:	<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
Allow Cookies	Enable cookies	Enabled/Disabled
Keep-Alive Enabled	Enable keep-alive connections	Enabled/Disabled
Authentication Scheme	HTTP authentication method	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous
Realm	The realm of the authentication scheme	Any URL
Require Client Certificate	For SSL transport, require a certificate	Enabled/Disabled

Proxy Information

If the Web service's transport uses a proxy server, you can specify its details in the **Security** tab. The following table describes the proxy options:

Option	Meaning	Possible Values
Use Default Web Proxy	Use machine's default proxy settings	Enabled/Disabled
Bypass Proxy on Local	Ignore proxy when the service is on the local machine	Enabled/Disabled

Option	Meaning	Possible Values
Proxy Address	the proxy server	Any URL
Proxy Authentication Scheme	HTTP authentication method on Proxy	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

Customizing Your Security Model

The built-in scenarios with the scenario editor allow you to test most Web Services that use advanced standards.

However, you may still need to manually change a configuration file.

To modify a configuration file:

- 1 Select or set up a scenario as described in "Setting the Security Scenario for a Service" on page 167.
- 2 Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
- 3 Navigate to the inner folder **%Script Root%/WSDL/@config**. This folder contains one or more .stss files.
- 4 Open the relevant .stss file. If the directory only contains one .stss file, open it with a text editor. If it contains more than one, open the **configurationIndex.xml** to determine which .stss file matches the service you currently try to configure.
- 5 Modify the configuration file as described in the sections below. After you modify the file, do not update the configuration again from the Manage Services **Protocol and Security** tab, as this will override the changes.

The following are the capabilities that can be achieved by using the configuration files.

Increasing Response Buffer Capacity

In cases where you expect a large response, as is common when using MTOM, replay may issue the following error:

Error: The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the `MaxReceivedMessageSize` property on the appropriate binding element.

To solve this, add the `maxReceivedMessageSize` attribute to the `httpTransport` element and configure it to use a larger size. For example:

```
<protocols scenario="customBinding xmlns="http://hp/ServiceTest/config">
  <customization>
    <mtomMessageEncoding />
    <httpTransport maxReceivedMessageSize="6000000" />
  </customization>
</protocols>
```

Customizing Windows Credentials

Some of the security scenarios use the Windows credentials. This is common with services utilizing WCF `WsHttpBinding` `SPNEGO` and `Kerberos`. By default, Service Test will use the currently logged in user as the client identity. You can instruct Service Test to use the identity of another user by modifying the appropriate configuration file. If any of the elements are already present in the configuration file, update them with the new information—do not duplicate the element.

```
<protocols ...>
  <identities>
    <client>
      <windowsCredentials>
        <username>myUser</username>
        <password>myPassword</password>
        <domain>myDomain</domain>
      </windowsCredentials>
    </client>
  </identities>
  ...
</protocols>
```

Using a Logical Address (**clientVia** Behavior)

This section describes how to specify a logical address instead of a physical one emulating **clientVia** behavior. In this case, you send a message to an intermediate service that submits it to the actual server. This may also happen when you send the message to a debugging proxy.

In such cases it may be useful to separate the physical address to which the message is actually sent, from the logical address for which the message is intended. The logical address may be the physical address of the final server or any name. It appears in the SOAP message as follows:

```
<wsa:Action>http://myLogicalAddress</wsa:Action>
```

Configuring the Logical Address

The logical address is determined in the Service Test user interface. By default, it is the address specified in the WSDL. You can override this address from the Manage Services dialog box.

Configuring the Physical Address

To set the physical address, you need to modify the configuration file. Under the **protocols** element, modify the **behaviors** element. Make sure to change the logical address to the correct one.

```
<protocols scenario="customBinding" xmlns="http://hp/ServiceTest/config">
...
<behaviors>
  <endpointBehaviors>
    <behavior>
      <clientVia viaUri="http://MyLogicalAddress" />
    </behavior>
  </endpointBehaviors>
</behaviors>
</protocols>
```

Note: When the above behavior is not present in the configuration file (as in the default behavior), then the logical address will also be the physical one, and requests will be sent to it.

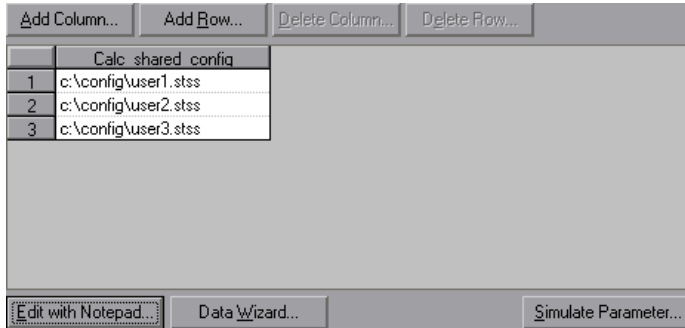
Parameterizing Security Elements

Scripts that utilize the new security model can be parameterized in the normal way. You can also parameterize the security elements independently. For example, in a username-based security scenario, you might want each Vuser or iteration to use a different user name.

To create parameters individually for each of the security elements:

- 1** Open the scenario editor. Select **SOA Tools > Security Scenario Editor**.
- 2** Set up and save a scenario for each Vuser. We recommend you use the names **user1**, **user2**, and so forth, and save them in a new folder, **%script root%/WSDL/referencedConfig**.
- 3** Open the Parameter List window. Select **Vuser > Parameters List**.
- 4** Create a new parameter, **<ServiceName>_shared_config**. Replace the **<ServiceName>** with the case-sensitive name of the service you are testing. To determine the exact name of the service, click **Manage Services** to see the list of services.

- In the parameters dialog box, add the file names of the security scenarios with their .stss extensions as parameter values. You can use a relative path, which is relative to the script root folder. Click **Add Row** to add multiple values.



- Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.
- Select **Shared Scenario**. Click the Browse button and enter the parameter name, <ServiceName>_shared_config, in the test box.

Simulating Users with Iterations

Many of the security scenarios establish a session with the server. For example, every scenario that uses **WS-SecureConversation** establishes a server session. This session is established when the first operation is executed and ends when the script is finished. By default, Service Test closes all sessions after each iteration and opens them again when the next iteration begins. This implies that every iteration simulates a new session and Vuser.

When working with multiple iterations, this may not be the desired effect—you may prefer to keep the original session active and not open a new session for each iteration. This applies when load testing through the LoadRunner Controller or when setting multiple iterations in the run-time settings.

You can override this behavior so that only the first iteration will establish a new session, while all subsequent ones will continue to use the open session. This simulates a user who repeatedly performs an action using the same session.

To determine which simulation mode to use, choose the one which is most appropriate to what you are simulating. For example, if you are simulating a load test where most of the actions are performed repeatedly by the same user in a single session, use the above configuration. If you are unsure, leave the default settings.

To configure your environment to use the same session for all iterations:

- 1** Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
- 2** Open **default.cfg** file in a text editor.
- 3** In the **[WebServices]** section, add in a row under the toolkit.

```
[WebServices]
Toolkit=.Net
SimulateNewUserInNewIteration=0
```

If you are using the Axis toolkit or if you configured other settings, the file contents may differ.

- 4** Save and close the file.

Tips and Guidelines

This section provides a quick summary of using Service Test for WCF, general security, and advanced standards testing.

WCF

How do I test a WCF service?

Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF** node and choose the relevant scenario according to its binding. If you could not find an appropriate binding, choose the **customBinding** scenario since it can test any other binding.

How do I test a WCF service that uses WSHttpBinding?

WSHttpBinding is one of the most popular bindings in WCF. In order to use this binding, click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF > By client authentication type** node and choose the client credential type that you use in your binding. This value corresponds to the **MessageClientCredentialType** property of the WCF's WSHttpBinding.

Windows authentication is the default value for a new WCF service. If you are using the WCF default settings for your service, use this option. Other options are username, certificate, or none. A username can be at the message level or at the transport level (equivalent to **TransportWithMessageCredential** in WCF).

For some scenarios you should indicate whether to use the WCF proprietary negotiation mechanism to get the service credentials.

Use the Advanced scenario properties to control the usage of a secure session.

How do I test a WCF service that uses CustomBinding?

Choose a scenario type of **WCF > Custom Binding** as described in "Scenario Types" on page 170. You can then customize many binding elements, such as your transport method, encoding, security, and reliable messaging.

How do I test a WCF service that uses netTcp or namedPipe transport?

Choose a scenario type of **WCF > Custom Binding** as described in "Scenario Types" on page 170. Configure the transport to **TCP** or **NamedPipe**.

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the `web_service_set_security_saml` function. For more information, see the *Online Function Reference (Help > Function Reference)* or Chapter 9, "Web Services - Security."

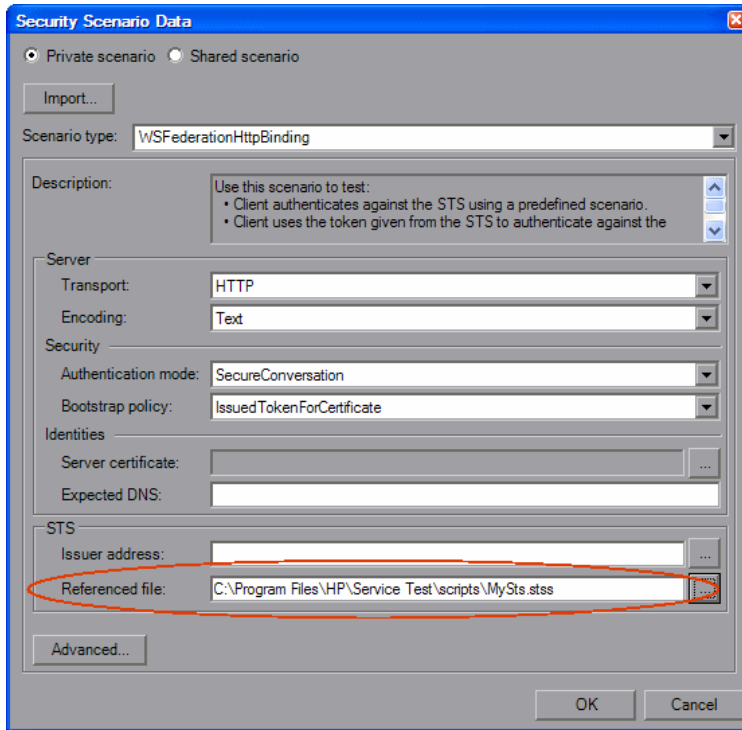
Choose the scenario **WCF > WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

- 1** Open the standalone security scenario editor. Select **SOA Tools > Security Scenario Editor**.
- 2** Click the **New** button. Configure the communication with the STS.
- 3** Click the **Save as** and specify a file name.

- 4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.



General Security

How do I test a Web Service that uses SSL?

Testing a secure site does not require any special configuration. If your service URL begins with **https**, SSL is automatically used. If in addition to SSL you are using message-level security (for example a username) then you must configure the security for the message separately using the legacy or the scenario-based model. If you use the scenario-based model, you need to configure it to use SSL by choosing an HTTPS transport or a transport credentials mode in a WSHttpBinding scenario.

How do I test a Web Service that require Windows authentication at the HTTP level?

Use the `web_set_user` function. If additional standards are required, use the Legacy security based model in conjunction with or instead of the scenario-based model.

How to I test a Web Service that uses WS-Security?

Use the scenario-based as described in this section or the legacy security using `web_service_set_security`.

How do I configure the low-level details of my WS-Security tokens?

In most cases, you can configure the low-level details as described in "Advanced Settings" on page 179. In case a very low-level control over the WS-Security tokens is required use the legacy security model. For more information, see Chapter 9, "Web Services - Security."

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the `web_service_set_security_saml` function. For more information, see the *Online Function Reference* (**Help > Function Reference**) or Chapter 9, "Web Services - Security."

Choose the scenario **WCF > WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

- 1** Open the standalone security scenario editor. Select **SOA Tools > Security Scenario Editor**.
- 2** Click the **New** button. Configure the communication with the STS.

- 3 Click the **Save as** and specify a file name.
- 4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.

General Protocols

How do I indicate not to use any advanced configurations?

As a scenario type, choose <no scenario>.

If you selected a scenario and during replay you receive errors, it is possible that you do not need an advanced scenario. Choose <no scenario> to cancel the existing selection and rerun the script.

How do I test a Web Service that uses MTOM?

Choose the **MTOM** scenario. If additional security is required, use one of the other scenarios. In the Advanced dialog box, set the encoding to MTOM. For more information, see "Advanced Settings" on page 179.

How do I change the WS-Addressing version of a service?

By default, the .NET toolkit uses WS-Addressing 2004/03, while the Axis toolkit does not use any addressing. To override this behavior, choose the **Plain SOAP** scenario and select the WS-Addressing version. Other supported versions are 2004/08, 1.0, and None. If your service requires additional standards, such as security, use the appropriate scenario and configure the addressing version from the **Encoding** tab in the Advanced window. For more information, see "Advanced Settings" on page 179.

11

Web Services - Transport Layers and Customizations

You can customize Web Service calls by setting the transport layer properties and by writing user handlers to define the behavior of the Web Service calls.

This chapter includes:

- About Testing Web Service Transport Layers on page 195
- Configuring the Transport Layer on page 196
- Sending Messages over HTTP/HTTPS on page 197
- Understanding JMS on page 198
- Sending Asynchronous Messages on page 202

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

About Testing Web Service Transport Layers

Web services can be sent over various transport layers. The transport layer is the protocol used to transport messages to and from the server.

VuGen allows you to configure the transport layer for your services. It fully supports HTTP/HTTPS and JMS (Java Message Service) transport layers.

The Service Test solution allows you to emulate both synchronous and asynchronous messaging. If you are working with HTTP/HTTPS transport, you can also use WS-Addressing.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see Chapter 12, "Web Services - User Handlers and Customization."

Configuring the Transport Layer

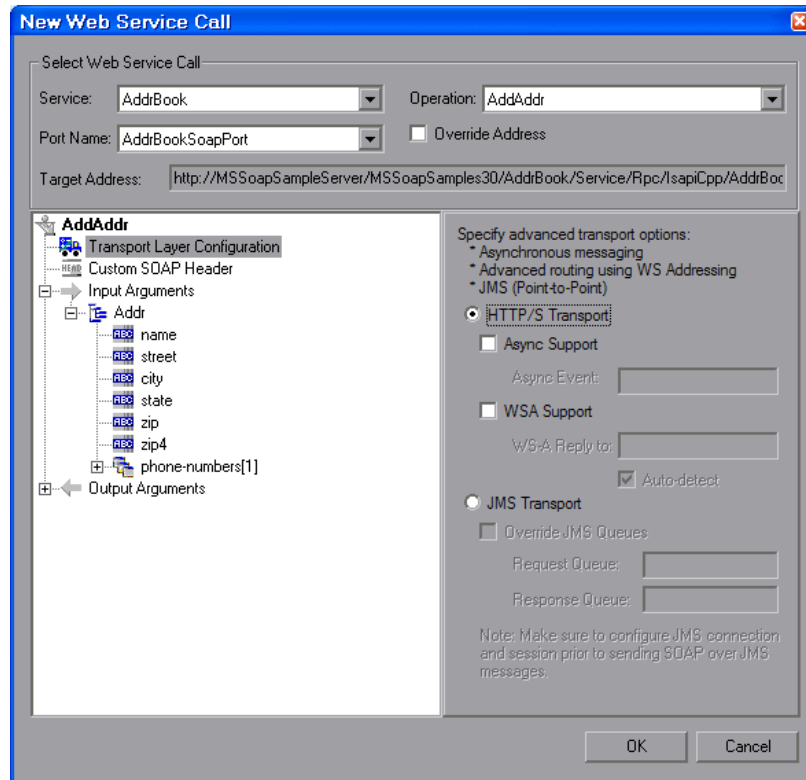
VuGen allows you to set the transport layer for your services. The transport layer indicates how to transport messages to and from the server—HTTP/HTTPS or JMS (Java Message Service).

For HTTP/HTTPS, see "Sending Messages over HTTP/HTTPS" on page 197. For more information about using the JMS transport, see "Understanding JMS" on page 198.

To choose a transport layer:

- 1 Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the Transport Layer Configuration node.



3 Choose the desired transport mode: HTTP/S Transport or JMS Transport.

For more options, see "Sending Messages over HTTP/HTTPS" below or "Using JMS as a Transport Layer" on page 199.

Sending Messages over HTTP/HTTPS

HTTP is used for sending requests from a Web client, usually a browser, to a Web server. HTTP is also used to return the Web content from the server back to the client.

HTTPS handles secure communication between a client and server. Typically, it handles credit card transactions and other sensitive data.

The typical request and response mechanism is synchronous. In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls in conjunction with WS-Addressing. For more information, see "Sending Asynchronous Calls with HTTP/HTTPS" on page 203.

Service Test also lets you emulate asynchronous messaging for your Web Service call. For more information, see "Sending Asynchronous Calls with JMS" on page 210.

Understanding JMS

JMS is a J2EE standard for sending messages, either text or Java objects, between Java clients.

There are two scenarios for communication:

Peer-to-Peer. Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.

Publish-Subscribe. Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

VuGen supports point-to-point communication by allowing you to send and receive JMS messages to and from a queue.

Before you can send messages over JMS transport, you need to configure several items that describe the transport:

- ▶ **JNDI initial context factory.** The class name of the factory class that creates an initial context which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- ▶ **JNDI provider.** The URL of the service provider which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- ▶ **JMS connection factory.** The JNDI name of the JMS connection factory.

In addition, you can set a timeout for received messages and the number of JMS connection per process.

You can configure all of these settings through the JMS run-time settings, as described in "Web Services JMS Run-Time Settings" on page 123.

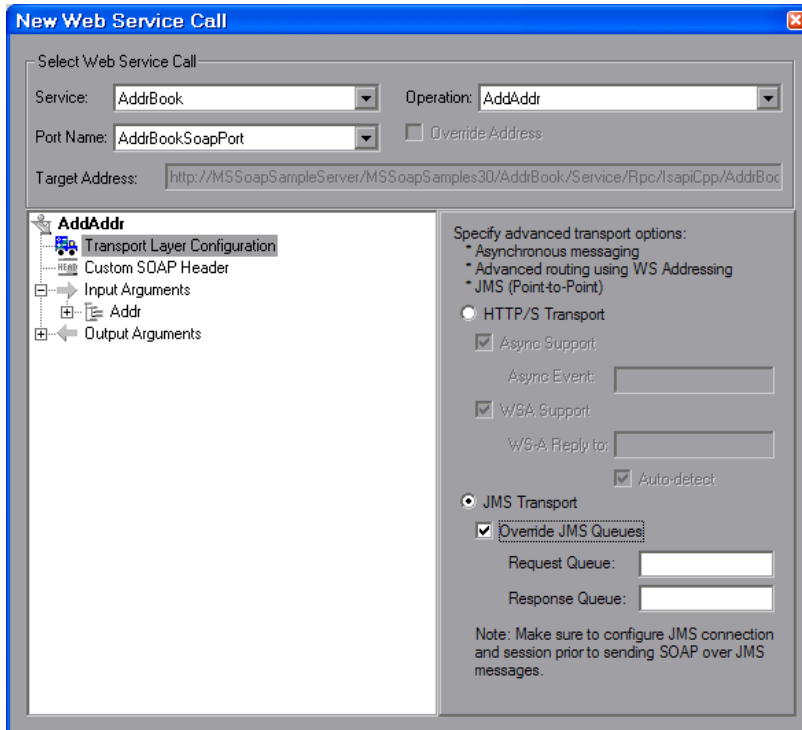
Using JMS as a Transport Layer

The JMS transport for Web services allows you to send SOAP messages using a JMS transport instead of the common HTTP transport.

To send messages using the JMS transport layer:

- 1** Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the **Transport Layer Configuration** node and select **JMS Transport**.



3 Configure the JMS run-time settings before running the script, as described in "Web Services JMS Run-Time Settings" on page 123.

The procedure above describes how to send synchronous JMS messages. For information on emulating asynchronous messages over JMS, see "Sending Asynchronous Calls with JMS" on page 210.

You can also send messages over JMS, even without creating Web Service calls. For example, using VuGen, you can:

- ▶ record SOAP messages using a standard Web protocol, and send them to a queue through JMS transport functions.
- ▶ manually send and receive general JMS messages from designated queues.

For more information, see "JMS Script Functions" below.

JMS Script Functions

VuGen uses its API functions to implement the JMS transport. Each function begins with a **jms** prefix:

Function Name	Description
jms_receive_message_queue	Receives a message from a queue
jms_send_message_queue	Sends a message to a queue.
jms_send_receive_message_queue	Sends a message to a specified queue and receives a message from a specified queue.
jms_set_general_property	Sets a general property in the user context.
jms_set_message_property	Sets a JMS header or property for the next message to be sent.

The JMS steps/functions are only available when manually creating scripts—you cannot record JMS messages sent between the client and server.

For additional information about the JMS functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

JMS Message Types

Each JMS message is composed of:

- **Header.** contains standard attributes (Correlation ID, Priority, Expiration date).
- **Properties.** custom attributes.
- **Body.** text or binary information.

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

Service Test attempts to resolve the desired format based on the message's content type. If the content type is **text/***, it sends the message in **TextMessage** format. Otherwise, it sends it in **BytesMessage** format.

To override the default behavior, use a `jms_set_general_property` function before sending the message. Set the `JMS_MESSAGE_TYPE` property to `TextMessage`, `BytesMessage`, or `Default`. For Example:

```
jms_set_general_property("step1","JMS_MESSAGE_TYPE","BytesMessage");
```

For more information, see the *Online Function Reference*.

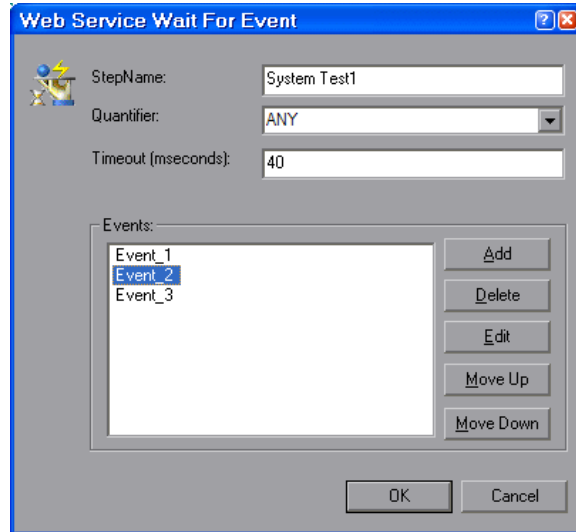
Sending Asynchronous Messages

You can use VuGen to emulate both synchronous and asynchronous messaging.

In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

Sending Asynchronous Calls with HTTP/HTTPS

This following section describes how to use asynchronous calls in HTTP/HTTPS. You use a **Wait for Event** step to instruct Vusers to wait for the response of previous asynchronous requests before continuing. The listener blocks the execution of the service until the server responds.



When adding a **Web Service Wait for Event** step:

- ▶ **Quantifier.** The quantifier indicates whether the Vuser should wait for **ALL** events to receive a response or **ANY**, just one of them. **ANY** returns the name of the first event to receive a response. **ALL** returns one of the event names.
- ▶ **Timeout.** the timeout in milliseconds. If no events receive responses in the specified timeout, then `web_service_wait_for_event` returns a NULL.
- ▶ **Events.** a list all of the asynchronous events for which you want to wait.

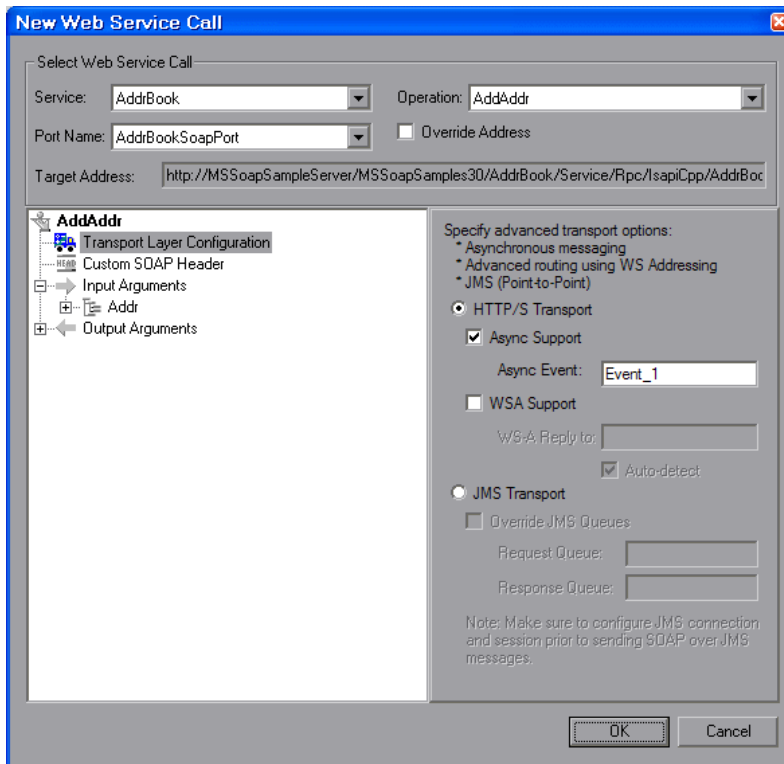
When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

For additional information about the **Web Service Wait for Event** step or `web_service_wait_for_event` function, see the *Online Function Reference* (**Help > Function Reference** or click **F1** on the function).

When setting up an asynchronous message, you can set the location to which the service responds when it detects an event using WS-Addressing. For more information, see "WS-Addressing" on page 206.

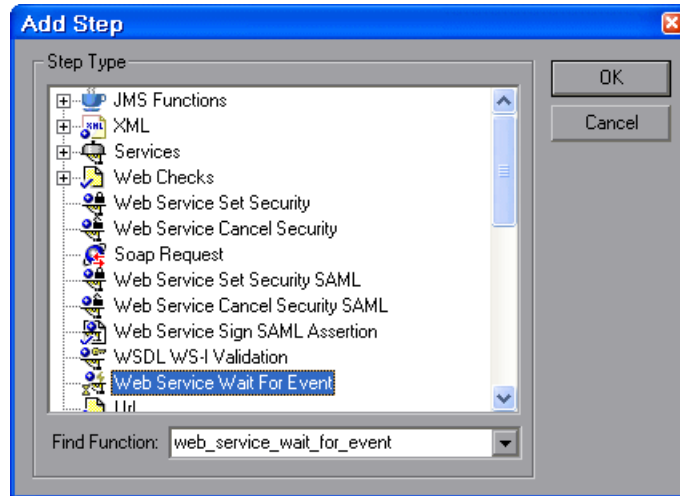
To send an asynchronous message using HTTP/S as a transport protocol:

- 1 In Tree view, select the **Step Properties** tab and select the **Transport Layer Configuration** node.
- 2 Choose **HTTP/S Transport** and select the **Async Support** option.

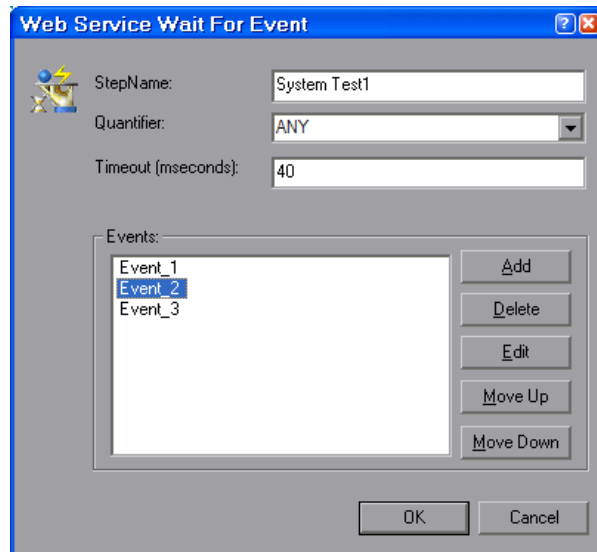


- 3 Type an event name in the **Async Event** box.
- 4 Click **OK** to generate the Web Service call.

- 5 Add a **Wait for Event** step. Select **Insert > New Step** and choose **Web Service Wait for Event**.



- 6 Specify a step name, a quantifier, and a timeout. Click **Add** and insert the name of the event that you defined in the previous step.



In Script view, VuGen indicates asynchronous messaging with the added parameter, **AsyncEvent**.

```
web_service_call("StepName=EchoString_101",
    "SOAPMethod=EchoRpcEncoded.EchoSoap.EchoString",
    "ResponseParam=response1",
    "Service=ExtendedECHO_rpc_encoded",
    "AsyncEvent=Event_1",
    "Snapshot=t1157371707.inf",
    BEGIN_ARGUMENTS,
    "sec=7",
    "strString=mytext",
    END_ARGUMENTS,
    BEGIN_RESULT,
    "EchoStringResult=first_call",
    END_RESULT,
    LAST);
```

The **AsyncEvent** flag instructs the Vuser to wait for the response of previous asynchronous service requests.

For synchronous messages, create a standard Web Service call, and do not enable the **Async Support** option.

WS-Addressing

WS-Addressing is a specification that allows Web Services to communicate addressing information. It does this by identifying Web service endpoints in order to secure end-to-end endpoint identification in messages. This allows you to transmit messages through networks that have additional processing nodes such as endpoint managers, firewalls, and gateways. WS-Addressing supports Web Services messages traveling over both synchronous or asynchronous transports.

The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

An optional **WSAAction** argument allows you to define a SOAP action for instances where transport layers fails to send a message.

The following example illustrates a typical SOAP message using WS-Addressing, implemented in the background by VuGen.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2004/12/addressing">
  <S:Header>
    <wsa:MessageID>
      http://example.com/SomeUniqueMessageIdString
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://myClient.example/someClientUser</wsa:Address>
    </wsa:ReplyTo>
    <wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
    </wsa:FaultTo>
    <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
    <wsa:Action>http://myserver.example/DoAction</wsa:Action>
  </S:Header>
  <S:Body>
    <!-- Body of SOAP request message -->
  </S:Body>
</S:Envelope>
```

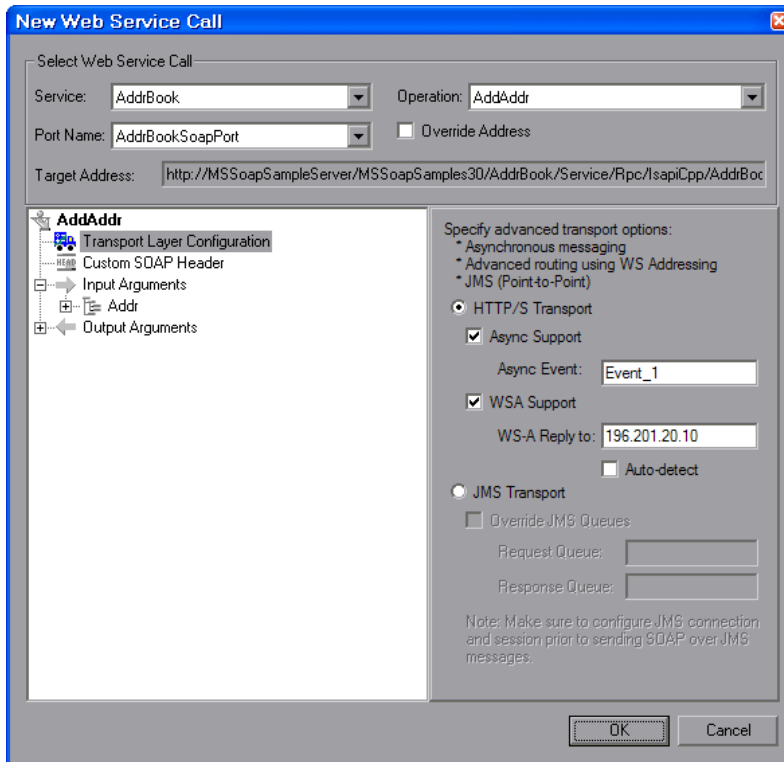
In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
web_service_call("StepName=Add_101",
  "SOAPMethod=Calc.CalcSoap.Add",
  "ResponseParam=response",
  "AsyncEvent=Event_1",
  "WSAReplyTo=212.199.95.138",
  "WSDL=http://lab1/WebServices/CalcWS/Calc.asmx?wsdl",
  "UseWSDLCopy=1",
  "Snapshot=t1153825715.inf",
  BEGIN_ARGUMENTS,
  "first=1",
  "second=2",
  END_ARGUMENTS,
  BEGIN_RESULT,
  "AddResult=Param_AddResult1",
  END_RESULT,
  LAST);
```

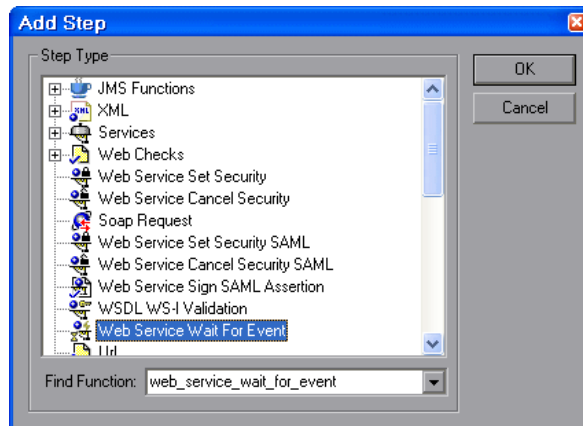
You can issue WS-Addressing calls in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, leave the **Async Event** box empty in the Transport Layer options. In Script view, remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

To add support for asynchronous messages and WS-Addressing:

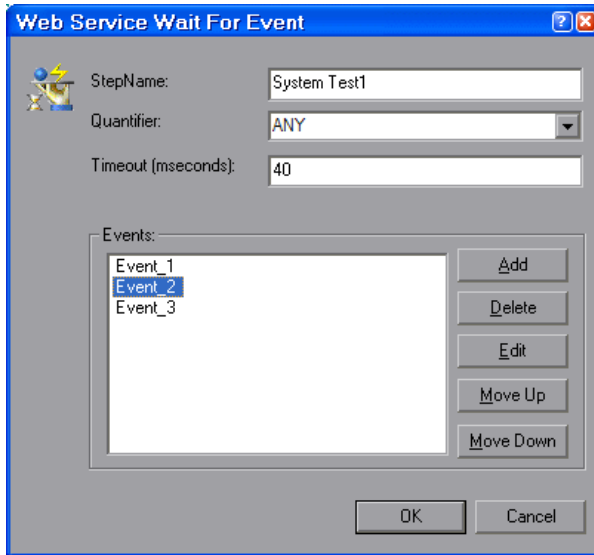
- 1 For a new Web Service call, select the **Transport Layer Configuration** node. For an existing Web Service call, select the step in Tree view, and select the **Step Properties** tab. Select the **Transport Layer Configuration** node.



- 2 To mark the Web Service call as an asynchronous message:
 - Select the **Async Support** option. This is only enabled for HTTP/S type transport.
 - Provide an event name in the **Async Event** box. This can be an arbitrary name.
- 3 Select **WSA Support**. In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. Autodetect is useful when running the same script on several different machines. The server will reply to the specified location when the event occurs.
- 4 Click **OK** to save the settings.
- 5 Instruct the Vuser to wait for an event. Select **Insert > New Step** and add a **Web Service Wait For Event** step after the Web Service call step.



- 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.



- Use the **Edit**, **Move Up**, and **Move Down** buttons to manipulate the events.
- Click **OK**.

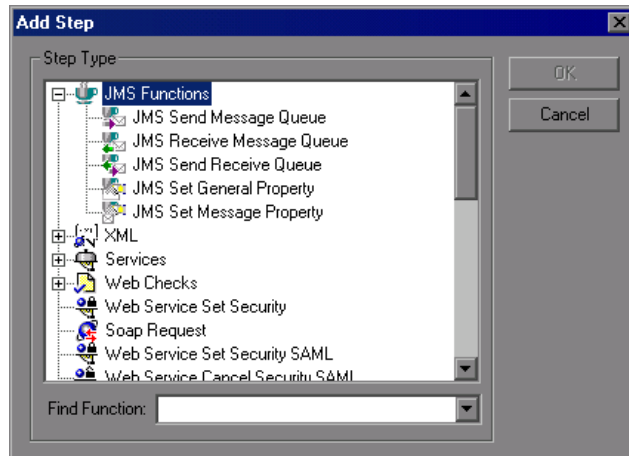
Sending Asynchronous Calls with JMS

This section describes how to send asynchronous messages using JMS.

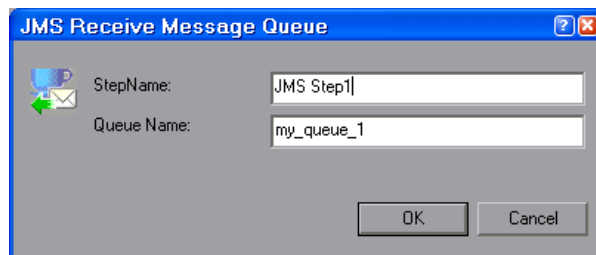
To implement this, you send the request or retrieve the response using JMS steps—not Web Service calls. **JMS Send Message Queue** sends a message to a queue. **JMS Receive Message Queue** receives a message from the queue.

To send and receive asynchronous messages using JMS:

- 1 Click within the script at the desired location. Select **Insert > New Step** and expand the **JMS Functions** node.



- 2 Select a JMS function to set up the message queue information: **Send Message Queue** or **JMS Receive Message Queue**.
- 3 Click **OK** to open the properties the JMS functions.



- 4 Specify a queue name and click **OK** to generate the JMS functions.

For JMS synchronous messages, create a standard Web Service call, select JMS transport, and if desired, specify the queue information.

For additional information about these functions, see the *Online Function Reference* (**Help > Function Reference** or click **F1** on the function).

12

Web Services - User Handlers and Customization

Service Test allows you to customize your script with user handlers and configuration files.

This chapter includes:

- ▶ About Customizing Web Service Script Behavior on page 213
- ▶ Setting up User Handlers on page 214
- ▶ User Handler Examples on page 219
- ▶ Using Custom Configuration Files on page 223

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

About Customizing Web Service Script Behavior

VuGen provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are user handlers and configuration files.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see below.

Configuration files let you customize advanced settings such as security information and the WSE configuration. For more information, see "Using Custom Configuration Files" on page 223.

Setting up User Handlers

User Handlers are open API through which you can perform the following operations:

- Get and set the request/response SOAP envelopes
- Override the transport layer
- Get and set the request/response content type
- Get and Set values for LoadRunner parameters
- Retrieve a configuration argument from the script
- Issue messages to the execution log
- Fail an execution

You can set up a user handler directly in a script, or you can implement through a DLL. You can apply a handler locally or globally. For details, see the following sections:

- ❑ Defining a Handler Function in a Script
- ❑ Creating a Custom User Handler as a DLL
- ❑ Configuring the User Handler
- ❑ Implementing the User Handler

For sample user handlers, see "User Handler Examples" on page 219.

Defining a Handler Function in a Script

For basic implementation of a user handler, you define a user handler function within your Vuser script:

```
int MyScriptFunction(const char* pArgs, int isRequest)
{
  ...
}
```

The **pArgs** argument contains the string that is specified in **UserHandlerArgs** argument of **web_service_call** function. For more information, see the *Online Function Reference (Help > Function Reference)*.

The **isRequest** argument indicates whether the function is being called during processing of a Request (1) or Response (0) SOAP envelope.

The content of SOAP envelope is passed to a parameter called **SoapEnvelopeParam** for both requests and responses. After the function processes the SOAP envelope, make sure to store it in the same parameter

To call the handler function, specify the function name as a value for the **UserHandlerFunction** argument in the relevant Web Service Call step.

```
web_service_call(
...
"UserHandlerFunction=MyScriptFunction",
"UserHandlerArgs=<handler arguments>",
LAST);
```

VuGen recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_SUCCEEDED_AND_MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in SoapEnvelopeParam .

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}

Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    return 0;
}
```


Creating a Custom User Handler as a DLL

You can also define a user handler by creating a DLL file through Visual Studio and the handler API. The API header file, **LrWsHandlerAPI.h**, located in the **LoadRunner/include** folder, contains many in-line comments and descriptions.

VuGen provides a sample Visual Studio project that can be used as a template for creating a handler. The sample retrieves the request and response envelope and saves it to a parameter. This sample is located in the **LoadRunner/samples/WebServices/SampleWsHandler** folder. To use this sample, open it in Visual Studio and modify it as required. If you do not need to save the request/response to a parameter, you can remove that section of the sample.

After editing the sample, save it and compile the DLL. When you compile the project, Visual Studio places the **<user_handler_name>.DLL** file in the **LoadRunner/bin** folder. If you compile the project from another location, or if you want to copy the DLL from one machine to another, make sure to place it in the bin folder.

Configuring the User Handler

You can declare the DLL user handler globally or locally.

To use the handler globally, for all requests in the script, add the following section to the **default.cfg** file located in the script's folder.

```
[UserHandler]
Function=<name>
Args=<arguments>
Order=<BeforeSecurity/AfterSecurity/AfterAttachments>
```

- **Name.** The name of the DLL.
- **Args.** A list of the configuration arguments for the handler. Use the **GetArguments** method to retrieve the arguments in your handler.
- **Order.** The order in which Vusers process the user handler in requests: **Before Security**, **After Security**, or **After Attachments**. You can also use this argument to override the transport layer, by entering the value **Replace Transport**.

Note: Setting the **UserHandlerFunction** property of a **web_service_call** function, overrides the definitions in the .cfg file.

By default, user handlers are processed before the security. For request messages, Vusers process the attachments handler after the security handler. For responses, Vusers process the handlers in a reversed order. In typical cases the order does not matter, so any value is acceptable.

To override the Transport layer, specify **Order=Replace Transport** and specify the new transport handler. If you implement the transport handler as a separate DLL, the **HandleRequest** function is called, while the **HandleResponse** function is ignored.

To use the handler locally, for a specific request, add the following arguments to the **web_service_call** function:

```
UserHandlerName=<name1>
UserHandlerArgs=<args1>
UserHandlerOrder=<BeforeSecurity/AfterSecurity/AfterAttachments/Replace
Transport>
```

Note: If you copy the script to another machine, it retains the handler information, since it is defined in script's folder. A user handler defined locally for a specific step in the script, overrides the global handler settings (defined in the script's **default.cfg** file).

Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the **LoadRunner/bin** folder.

Implementing the User Handler

To implement a user handler, you use the entry functions **HandleRequest** or **HandleResponse**. Both functions have a single parameter, **context**, whose properties you can set in your handler. Use the Get functions to retrieve properties, and Set functions to pass information from the replay framework to the handlers or between the handlers.

- **GetEnvelope**. Gets the envelope content. For example, example:

```
const char * pEnvelope = context->GetEnvelope();
```
- **GetEnvelopeLength**. Gets the envelope length
- **SetEnvelope**. Sets the envelope content and length. For example:

```
string str("MySoapEnvelope...");  
context->SetEnvelope(str.c_str(), str.length());
```
- **SetContentType**. Sets a new value for HTTP header content type
- **LogMessage**. Issues a message to the replay log
- **GetArguments**. Gets the configuration arguments defined for the current handler in order to pass it to the DLL
- **GetProperty**. Gets a custom property value
- **SetProperty**. Sets a custom property value

For more information, see the comments in the **LrWsHandlerAPI.h** file located in the **LoadRunner/include** folder.

User Handler Examples

The following section describes how to create user handlers for several common issues:

- .NET Filters
- Overriding the Transport Layer
- Including MIME Attachments

.NET Filters

If you are familiar with Microsoft's Web Service Enhancements (WSE) 2.0, you can create a .NET filter and register it for incoming or outgoing SOAP messages. A .NET filter is a class that is derived from `Microsoft.Web.Services2.SoapInputFilter` or `Microsoft.Web.Services2.SoapOutputFilter`. By overriding the **ProcessMessage** function of this class, you can examine and modify the envelope's body and header.

You can apply a .NET filter to your messages using the user handler mechanism.

To define the filter globally for the entire script, add the following lines to the script's default.config file below.

```
[UserHandler]
Function=LrWsSoapFilterLoader
Args=<Filters InputFilterClass="class name" InputFilterLib="lib name"
OutputFilterClass="class name" OutputFilterLib="lib name" />
Order=BeforeSecurity/AfterSecurity/AfterAttachments
```

The **InputFilterClass** parameter indicates the name of your class, and **InputFilterLib** indicates the name of the assembly in which the class resides. For example:

```
web_service_call(
...
  "UserHandlerName=LrWsSoapFilterLoader",
  "UserHandlerArgs=<Filters
InputFilterClass=\"MyFilterNamespace.MyFilterClassName\"
InputFilterLib=\"MyAssemblyName\" />",
  BEGIN_ARGUMENTS,
  ...
  END_ARGUMENTS,
  ...
);
```

Use `SoapOutputFilter` to examine an outgoing `web_service_call` request, and `SoapInputFilter` to examine the response from the server. Use **`InputFilterClass`** and **`InputFilterLib`** if your filter is derived from `SoapInputFilter`, or **`OutputFilterClass`** and **`OutputFilterLib`** if your filter is derived from `SoapOutputFilter`.

To define the filter for a specific step, add the following arguments to the `web_service_call` function.

```
UserHandlerName= LrWsSoapFilterLoader
UserHandlerArgs=<Filters InputFilterClass=\"class name\" InputFilterLib=\"lib name\"
OutputFilterClass=\"class name\" OutputFilterLib=\"lib name\" />
UserHandlerOrder=BeforeSecurity/AfterSecurity/AfterAttachments
```

Overriding the Transport Layer

You can write a user handler function to override the transport layer. In this case, VuGen will not automatically send the SOAP request over HTTP transport—instead it follows the transport method indicated in the custom handler.

After you receive a response, you can set the response envelope with the command:

```
lr_save_string(someResponseEnvelopeStr, "SoapEnvelopeParam");
```

To apply an alternate transport layer, specify **`ReplaceTransport`** as a value for the **`UserHandlerOrder`** argument, and define the transport layer in the handler function.

```
web_service_call(
...
"UserHandlerFunction=<Transport HandlerFunction>",
"UserHandlerArgs=<handler arguments>",
"UserHandlerOrder=ReplaceTransport"
...
LAST);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the `web_service_call`:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
  "UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  "ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
  "<cls>"
  "  <i>123456789</i>"
  "  <s>abcde</s>"
  "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
  LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the file you want to send and its content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the `web_service_call`:

```
"UserHandlerName=LrWsAttachmentsHandler",
"UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same `web_service_call`, add the following code:

```
"UserHandlerName=LrWsAttachmentsHandler",
"UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;
FileName=C:\\temp\\results.discomap",
```

Using Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration.

The standard .NET configuration file, `mmdrv.exe.config`, is located in the `LoadRunner/bin` folder.

If your application has its own configuration file, `app.config`, you can implement it in several ways:

- ▶ Save it as `mmdrv.exe.config`, overwriting the existing configuration file. This will apply your configuration information to all scripts on the machine.
- ▶ Save `app.config` to the script's folder. The settings in the `app.config` file override the ones in `mmdrv.exe.config`. In addition, if you save it to the script's file, it will always be associated with the script, not requiring you to copy it over separately to other machines.

Use the filter with the **Input** prefix if your filter is derived from SOAP input, or the **Output** prefix if your filter is derived from SOAP output.

In addition, the configuration file contains security information. You can configure whether or not to allow unsigned test certificates.

By default, VuGen allows unsigned certificates to facilitate testing. To disallow unsigned certificates, modify the **allowTestRoot** flag in the **mmdrv.exe.config** file to false.

```
<security>  
  <x509 storeLocation="currentuser" allowTestRoot="false"
```


13

Web Services - Negative Testing

Using VuGen, you can test your Web Service using different testing methodologies, such as positive or negative testing.

This chapter includes:

- ▶ About Applying Testing Methodologies on page 225
- ▶ Understanding the Testing Settings on page 226
- ▶ Defining a Testing Method on page 227
- ▶ Evaluating the SOAP Fault Value on page 229

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

About Applying Testing Methodologies

When performing a functional test for your Web Service, you should approach the testing in a variety of ways. The most common type of testing is called **Positive Testing**—checking that the service does what it was designed to do.

In addition, you should perform **Negative Testing**, to confirm that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued an appropriate error—a SOAP Fault.

To illustrate this, consider a form accepting input data—you apply positive testing to check that your Web Service has properly accepted the name and other input data. You apply negative testing to make sure that the application detects an invalid character, for example a letter character in a telephone number.

When your service send requests to the server, the server responds in one of the following ways:

- ▶ **SOAP Result.** A SOAP response to the request
- ▶ **SOAP Fault.** A response indicating that the SOAP request was invalid. Negative Testing applies only to SOAP faults.
- ▶ **HTTP Error.** An HTTP error, such as Page Not Found, unrelated to Web Services.

VuGencan check for a standard SOAP result or a SOAP fault responses—it always fails on HTTP errors. For example, if your Web Service attempts to access a Web page that cannot be found, resulting in a 404 HTTP error, the replay will fail, even if the SOAP is valid.

Understanding the Testing Settings

When creating a Web Service Call or SOAP Request in VuGen, you can indicate the type of testing you want to perform during replay:

Type of Testing	Description
Positive Testing	Accept SOAP result responses and fail on SOAP faults.
Negative Testing	Accept SOAP faults and fail on SOAP result responses.
Any Type	Accept both SOAP result and SOAP fault responses.

By default, VuGen, only performs positive testing and passes a test when it receives a SOAP result response. You can instruct VuGento perform only negative testing, or to accept any SOAP response. If you enable negative testing only, and the server issues a regular SOAP result response, the step will have a **Failed** status.

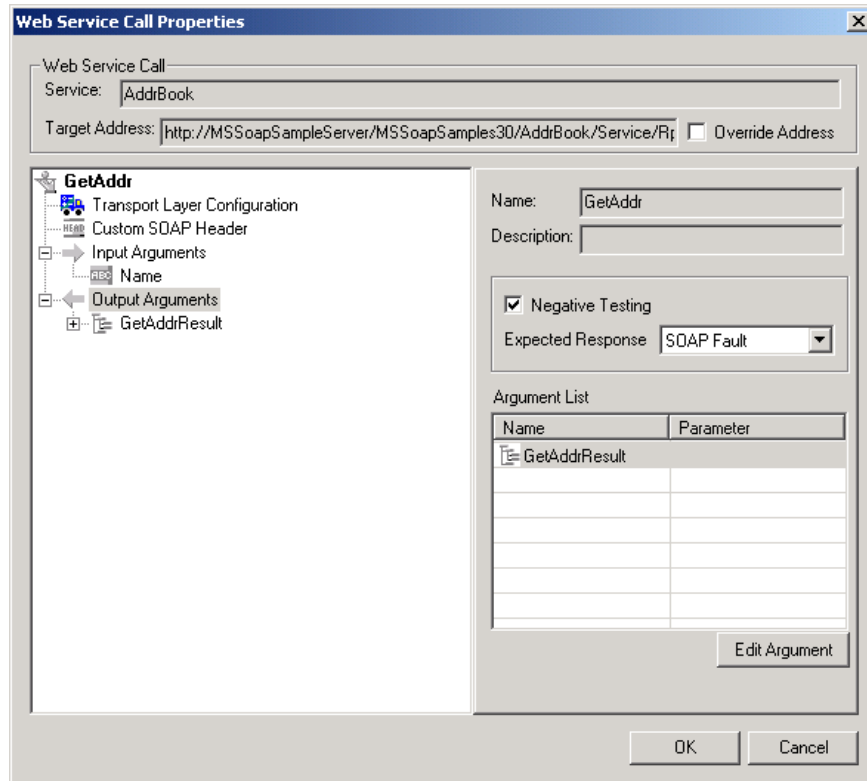
You can instruct VuGento accept any SOAP response—a SOAP result or SOAP fault. This can be useful in testing environments where you only need to send the request, using a separate function to check the SOAP at a later time. In this testing mode, steps with either a SOAP result or SOAP fault will be issued a **Passed** status.

You can check the status of the replay by viewing the Replay log and Test Results report. A failed step will be marked in red as **Failed** in the Test Results.

If you are working with Quality Center or HP Service Test Manager, the application will list the test's status based on the Expected Response setting.

Defining a Testing Method

Before replaying the script, you indicate the testing method in the Web Service Call properties dialog box—positive or negative testing, or any SOAP.



To select a testing method:

- 1** Select the step whose response you want to test. Open the Properties from the right-click menu.
- 2** Select the **Output Argument** node.
- 3** Choose an Expected Response.
 - To perform positive testing only, clear the **Negative Testing** check box.
 - To perform negative testing only, select the **Negative Testing** check box and choose **SOAP Fault** as the **Expected Response**.
 - To accept any type of SOAP response, select the **Negative Testing** check box and choose **Any SOAP** as the **Expected Response**.

In Script view, VuGen represents the testing method with the **ExpectedResponse** argument. The possible values are **SoapResult**, **SoapFault**, or **AnySoap**. In the following example, the script performs negative testing, indicated by the **SoapFault** value:

```
web_service_call("StepName=AddAddr_101",
    "SOAPMethod=AddrBook|AddrBookSoapPort|AddAddr",
    "ResponseParam=response",
    "Service=AddrBook",
    "ExpectedResponse=SoapFault",
    "Snapshot=t1189409011.inf",
    BEGIN_ARGUMENTS,
    END_ARGUMENTS,
    BEGIN_RESULT,
    END_RESULT,
    LAST);
```

Evaluating the SOAP Fault Value

When you replay a script that results in a SOAP fault, VuGen saves the fault to a parameter called **response**. To check the value of the SOAP fault that was returned, you evaluate the **response** output parameter using **lr_xml_find**.

In the following example, **lr_xml_find** checks for a **VersionMismatch** SOAP fault and issues an output message.

```
lr_xml_find("XML={response}",
           "FastQuery=/Envelope/Body/Fault/faultString ",
           "Value=VersionMismatch",
           LAST);

if (soap_fault_cnt >0)
    lr_output_message("A Version Mismatch SOAP Fault occurred")
```

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

14

Java Protocols - Recording

VuGen allows you to record applications or applets written in Java, in protocols such as CORBA, RMI, EJB, JMS or Jacada. You can also use VuGen's navigation tool to add any method to your script.

This chapter includes:

- About Recording Java Language Vuser Scripts on page 232
- Getting Started with Java Vuser Scripts on page 233
- Recording Java Events on page 235
- Recording CORBA on page 238
- Recording RMI over IIOP on page 239
- Recording RMI on page 240
- Recording a Jacada Vuser on page 240
- Recording on Windows XP and Windows 2000 Servers on page 241

About Recording Java Language Vuser Scripts

Using VuGen, you can record a Java application or applet. VuGen creates a pure Java script enhanced with Vuser API Java-specific functions. After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it. Once you verify that the script is functional, you incorporate it into a LoadRunner scenario or Business Process Monitor profile.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. See Chapter 27, "Programming Java Scripts" for important information about function syntax and system configuration.

Before recording a CORBA session, verify that your application or applet functions properly on the recording machine.

Make sure that you have properly installed a JDK version from Sun on the machine running VuGen—JRE alone is insufficient. You must complete this installation before recording a script. Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions.

Note: When you load an applet or application from VuGen during recording, it may take several seconds longer than if you were to load it independent of VuGen.

VuGen provides a tool that enables you to convert a Vuser script created for Web, into Java. For more information, see "Converting Web Vuser Scripts into Java" on page 560.

After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it.

You integrate finished scripts into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller, Performance Center*, or *HP Business Availability Center* documentation.

Getting Started with Java Vuser Scripts

The following procedure outlines how to record Java language Vuser scripts.

1 Make sure that the recording machine is properly configured.

Make sure that your machine is configured properly for Java before you begin recording. For more information, see Chapter 27, "Programming Java Scripts" and the Readme file.

2 Create a new Vuser script.

Select The Java Record/Replay Vuser type.

3 Specify a Java protocol.

Select a protocol from the recording options.

4 Set the recording parameters and options for the script.

You specify the parameters for your applet or application such as working directory and paths. You can also set JVM, serialization, correlation, recorder, and debug recording options. See Chapter 18, "Java Recording Options" in *Volume I-Using VuGen* for more information.

5 Record typical user actions.

Begin recording a script. Perform typical actions within your applet or application. VuGen records your actions and generates a Vuser script.

6 Enhance the Vuser script.

Add Vuser API specific functions to enhance the Vuser script. For details, see Chapter 27, "Programming Java Scripts." You can use the built-in Java function Navigator. For more information, see "Viewing the Java Methods" on page 250.

7 Parameterize the Vuser script.

Replace recorded constants with parameters. You can parameterize complete strings or parts of a string. Note that you can define more than one parameter for functions with multiple arguments. For details, see "Defining Parameters" in *Volume I-Using VuGen*.

8 Configure the run-time setting for the script.

Configure run-time settings for the Vuser script. The run-time settings define the run-time aspects of the script execution. For the specific run-time settings for Java, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

9 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

For detailed information on the recording procedure, see the specific chapter for your Vuser type.

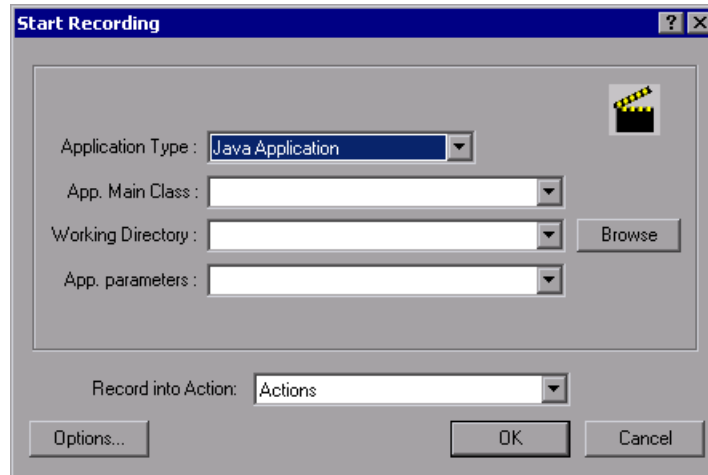
Recording Java Events

Make sure that you have properly installed a JDK version from Sun on the machine running the Vusers—JRE alone is insufficient. Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions. Before you replay a Vuser script, verify that your environment is configured properly for the JDK and relevant Java classes.

This is the general procedure for recording a Java session.

To begin recording:

- 1 Select **File > New** and select **JAVA Record Replay** from the **Java** category. The Start Recording dialog box opens.



- 2 In the **Application Type** box, select the appropriate value.
 - **Java Applet** to record a Java applet through Sun's appletviewer.
 - **Java Application** to record a Java application.
 - **Netscape** or **IEExplore** to record an applet within a browser.
 - **Executable/Batch** to record an applet or application that is launched from within a batch file or the name of an executable file.

- **Listener** to instruct VuGen to wait for the batch file that initializes the configuration and runs an application before recording. This mode requires you to define the system variable `_JAVA_OPTIONS` as `--Xrunjdkhook` using `jdk1.2.x` and higher. (For JDK 1.1.x, define the environment variable `_classload_hook=JDKhook`. For JDK 1.6 set `_JAVA_OPTIONS` as `-agentlib:jdhook`.)

3 Specify additional parameters according for the following chart:

Application Type	Fields to Set
Java Applet	Applet Path, Working Directory
Java Application	App. Main Class, Working Directory, App. parameters
IExplore	IExplore Path, URL
Netscape	Netscape Path, URL
Executable/Batch	Executable/Batch, Working Directory
Listener	N/A

Note: A Working Directory is necessary only if your application must know the location of the working directory (for example, reading property files or writing log files).

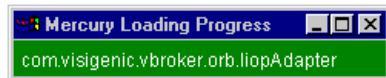
- 4** Click **Options** to open the Recording Options dialog box. You select a Java protocol: CORBA, RMI, JMS, or Jacada and set other recording properties. See Chapter 18, "Java Recording Options" in *Volume I-Using VuGen* for more information.

- 5 In the Record into Action box, select the section corresponding to the method into which you want to record. The Actions class contains three methods: `init`, `action`, and `end`, corresponding to the `vuser_init`, `Actions`, and `vuser_end` sections. The following table shows what to include into each method, and when each method is executed.

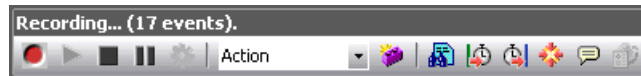
method within Actions class	Record into action	Used to emulate...	Executed during...
<code>init</code>	<code>vuser_init</code>	a login to a server	Initialization
<code>action</code>	<code>Actions</code>	client activity	Running
<code>end</code>	<code>vuser_end</code>	a log off procedure	Finish or Stopped

Note: Make sure to import the `org.omg.CORBA.ORB` function in the `vuser_init` section, so that it will not be repeated for each iteration.

- 6 Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.



- 7 Perform typical actions within your application. Use the floating toolbar to switch methods during recording.



- 8 After recording the typical user actions, select the **vuser_end** method from the floating toolbar.



Perform the log off procedure. VuGen records the procedure into the **vuser_end** method of the script.



- 9 Click **Stop Recording** on the Recording toolbar. The VuGen script editor displays all the recorded statements.



Click **Save** and provide a name for the script.

Recording CORBA

For recording a CORBA session, you need to set the following options in the Recording Options:

- JNDI
- Use DLL hooking to attach VuGen support

Using CORBA Application Vendor Classes

Running CORBA applications with JDK1.2 or later, might load the JDK internal CORBA classes instead of the specific vendor CORBA classes. To force the virtual machine to use the vendor classes, specify the following java.exe command-line parameters:

Visigenic 3.4

```
-Dorg.omg.CORBA.ORBClass=com.visigenic.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.visigenic.vbroker.orb.
  ORBSingleton
```

Visigenic 4.0

```
-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton
```

OrbixWeb 3.x

```
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.
  singletonORB
```

OrbixWeb 2000

```
-Dorg.omg.CORBA.ORBClass=com.ionacorba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.ionacorba.art.artimpl.
  ORBSingleton
```

Recording RMI over IIOP

The **Internet Inter-ORB Protocol** (IIOP) technology was developed to allow implementation of CORBA solutions over the World Wide Web. IIOP lets browsers and servers exchange complex objects such as arrays, unlike HTTP, which only supports transmission of text.

RMI over IIOP technology makes it possible for a single client to access services which were only accessible from either RMI or CORBA clients in the past. This technology is a hybrid of the JRMP protocol used with RMI and IIOP used with CORBA. **RMI over IIOP** allows CORBA clients to access new technologies such as **Enterprise Java Beans** (EJB) among other J2EE standards.

VuGen provides full support for recording and replaying Vusers using the **RMI over IIOP** protocol. Depending on what you are recording, you can utilize VuGen's RMI recorder to create a script that will optimally emulate a real user:

- ▶ **Pure RMI client.** recording a client that uses native JRMP protocol for remote invocations
- ▶ **RMI over IIOP client.** recording a client application that was compiled using the IIOP protocol instead of JRMP (for compatibility with CORBA servers).

Recording RMI

Before recording an RMI session, verify that your application or applet functions properly on the recording machine.

Before you record, verify that your environment is configured properly. Make sure that the required classes are in the classpath and that you have a full installation of JDK. For more information on the required environment settings, see Chapter 27, "Programming Java Scripts."

Recording a Jacada Vuser

The Jacada Interface Server provides an interface layer for mainframe applications. This layer separates the user interface from the application logic in order to insulate the organization from changes in standards and technologies. Instead of working with green-screen applications, the Jacada server converts the environment to a user friendly interface.

VuGen records Jacada's Java thin-client. To record communication with the Jacada server through the HTML thin-client, use the Web HTTP/HTML type Vuser. For more information, see Chapter 37, "Web (HTTP/HTML, Click and Script) Protocols."

Before replay, you must also download the **clbase.jar** file from the Jacada server. All classes used by the Java Vuser must be in the classpath—either set in the machine's CLASSPATH environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

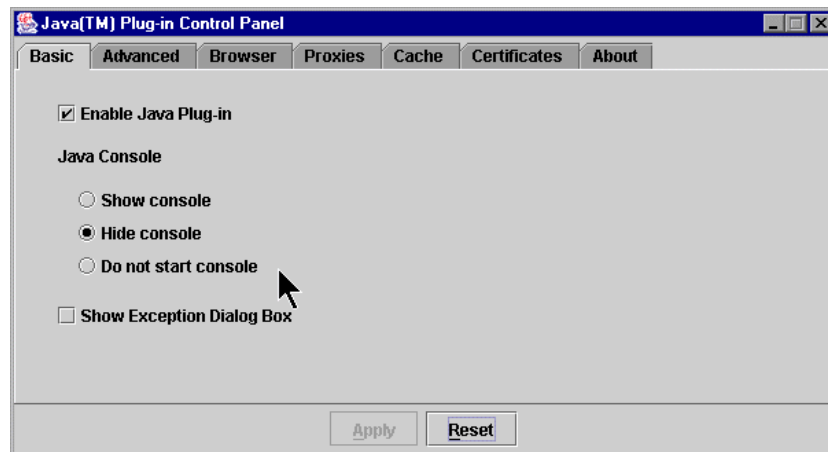
During replay, the Jacada server may return screens from the legacy system, in a different order than they appear in the recorded script. This may cause an exception in the replay. For information on how to handle these exceptions, please contact support.

Recording on Windows XP and Windows 2000 Servers

When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

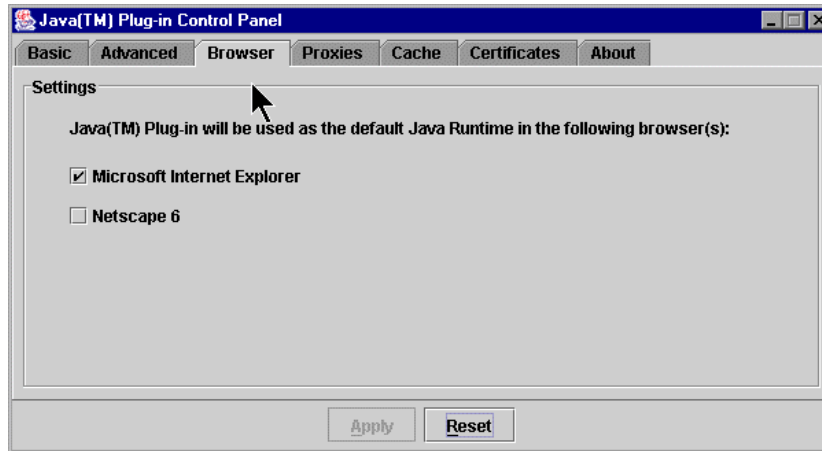
To configure your machine for recording CORBA or RMI sessions:

- 1 Open the Java Plug-in from the Control Panel. Select **Start > Settings > Control Panel** and open the **Java Plug-in** component. The Basic tab opens.



- 2 Clear the **Enable Java Plug-In** check box and click **Apply**. Then, reselect the **Enable Java Plug-In** check box and click **Apply**.

3 Open the Browser tab.



4 Clear the **Microsoft Internet Explorer** check box and click **Apply**. Then, reselect the **Microsoft Internet Explorer** check box and click **Apply**.

15

Java - Managing Vuser Scripts

VuGen allows you to record applications or applets written in Java. You can run the recorded script or enhance it using standard Java library functions and Vuser API Java-specific functions.

This chapter includes:

- Understanding Java Vuser Scripts on page 243
- Working with CORBA on page 245
- Working with RMI on page 247
- Working with Jacada on page 248
- Running a Script as Part of a Package on page 249
- Viewing the Java Methods on page 250
- Manually Inserting Java Methods on page 252
- Configuring Script Generation Settings on page 254
- Java Custom Filters on page 258

Understanding Java Vuser Scripts

When you record a session, VuGen logs all calls to the server and generates a script with functions. These functions describe all of your actions within the application or applet. The script also contains supplementary code required for proper playback, such as property settings, and naming service initialization (JNDI).

The recorded script is comprised of three sections:

- Imports

- ▶ Code
- ▶ Variables

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script. The **Code** section contains the Actions class and the recorded code within the **init**, **actions**, and **end** methods. The **Variables** section, after the **end** method, contains all the type declarations for the variables used in the code.

After you finish recording, you can modify the functions in your script, or add additional Java or LoadRunner functions to enhance the script. Note that if you intend to run Java Vusers as threads, the Java code you add to your script must be thread-safe. For details about function syntax, see the *Online Function Reference (Help > Function Reference)*. In addition, you can modify your script to enable it to run as part of another package. For more information, see "Compiling and Running a Script as Part of a Package" on page 440.

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 "Using the Serialization Mechanism" on page 272.

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 27, "Programming Java Scripts."

Running a Script as Part of a Package

This section is not relevant for Jacada type scripts.

When creating or recording a Java script, you may need to use methods from classes in which the method or class is protected. When attempting to compile such a script, you receive compilation errors indicating that the methods are not accessible.

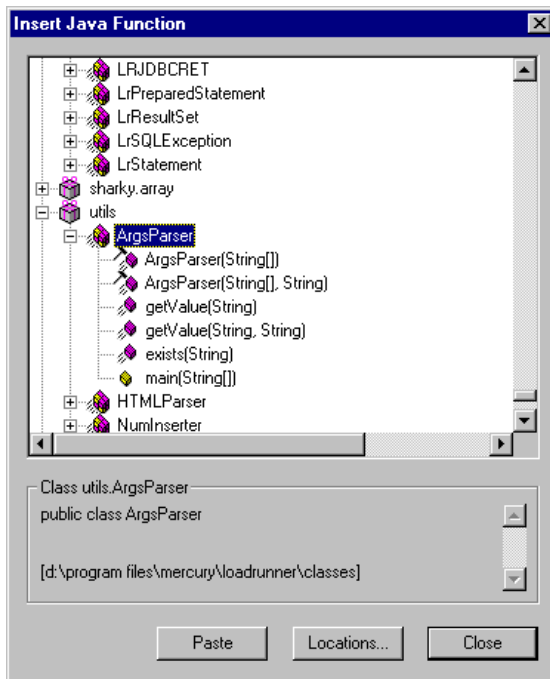
To use the protected methods, add the Vuser to the package of required methods. At the beginning of your script, add the following line:

```
package a.b.c;
```

where **a.b.c** represents a directory hierarchy. VuGen creates the *a/b/c* directory hierarchy in the user directory and compiles the **Actions.java** file there, thus making it part of the package. Note that the **package** statement is not recorded—you need to insert it manually.

Viewing the Java Methods

VuGen provides a navigator that lets you view all of the Java classes and methods in your application's packages.









To insert a class or method into your script, you select it and paste it into your script. For step-by-step instructions, see "Manually Inserting Java Methods" on page 252.

The lower part of the dialog box displays a description of the Java object, its prototype, return values and path. In the following example, the description indicates that the deserialize method is a public static method that receives two parameters—a string and an integer. It returns a java.lang.Object and throws an exception.

```
public static synchronized java.lang.Object deserialize (java.lang.String, int) throws
Exception
```

The following table describes the icons that represent the various Java objects:

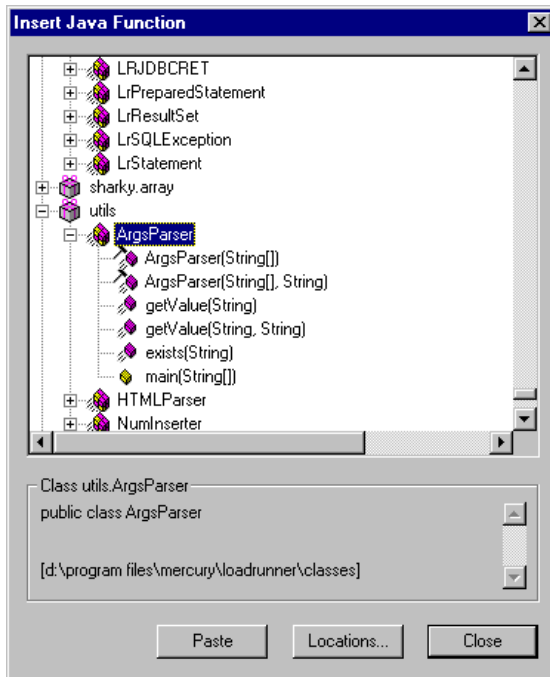
Icon	Item	Example
	Package	java.util
	Class	public class Hashtable extends java.util.Dictionary implements java.lang.Cloneable, java.io.Serializable
	Interface Class (gray icon)	public interface Enumeration
	Method	public synchronized java.util.Enumeration keys ()
	Static Method (yellow icon)	public static synchronized java.util.TimeZone getTimeZone
	Constructor Method	public void Hashtable ()

Manually Inserting Java Methods

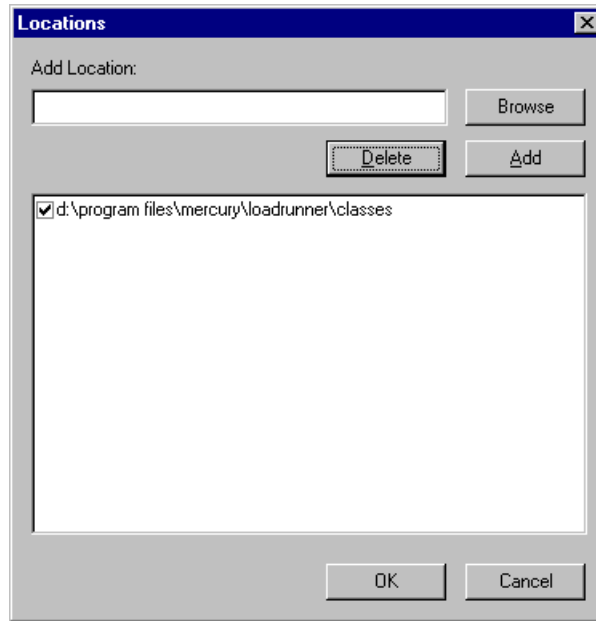
You use the Java Function navigator to view and add Java functions to your script. The following section apply to EJB Testing and Java Record/Replay Vusers. You can customize the function generation settings by modifying the configuration file. For more information, see "Configuring Script Generation Settings" on page 254.

To insert Java functions:

- 1 Click within your script at the desired point of insertion. When you paste a function, VuGen places it at the location of the cursor.
- 2 Select **Insert > Insert Java Function**. The Insert Java Function dialog box opens.



- 3** Click **Locations**. The Locations dialog box opens. By default, VuGen lists the paths defined in the CLASSPATH environment variable.



- 4** Click **Browse** to add another path or archive to the list. To add a path, select **Browse > Folder**. To add an archive (**jar** or **zip**), select **Browse > File**. When you select a folder or a file, VuGen inserts it in the **Add Location** box.
- 5** Click **Add** to add the item to the list.
- 6** Repeat steps 4 and 5 for each path or archive you want to add.
- 7** Select or clear the check boxes to the left of each item in the list. If an item is checked, its members will be listed in the Java Class navigator.
- 8** Click **OK** to close the Locations dialog box and view the available packages.
- 9** Click the plus and minus signs to the left of each item in the navigator, to expand or collapse the trees.
- 10** Select an object and click **Paste**. VuGen places the object at the location of the cursor in the script. To paste all the methods of a class into your script, select the class and click **Paste**.

- 11** Repeat the previous step for all of the desired methods or classes.
- 12** Modify the parameters of the methods. If the script generation setting **DefaultValues** is set to **true**, you can use the default values inserted by VuGen. If **DefaultValues** is set to **false**, you must add parameters for all methods you insert into the script.

In addition, modify any return values. For example, if your script generated the following statement "(String)=LavaVersion.getVersionId();", replace (String) with a string type variable.
- 13** Add any necessary statements to your script such as imports or Vuser API Java functions (described in Chapter 27, "Programming Java Scripts").
- 14** Save the script and run it from VuGen.

Configuring Script Generation Settings

You can customize the way the navigator adds methods to your script in the following areas:

- Class Name Path
- Automatic Transactions
- Default Parameter Values
- Class Pasting

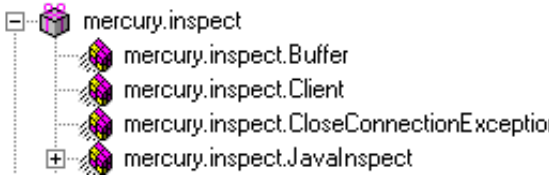

To view the configuration setting, open the **jquery.ini** file in VuGen's dat directory.

```
[Display]
FullClassName=False

[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

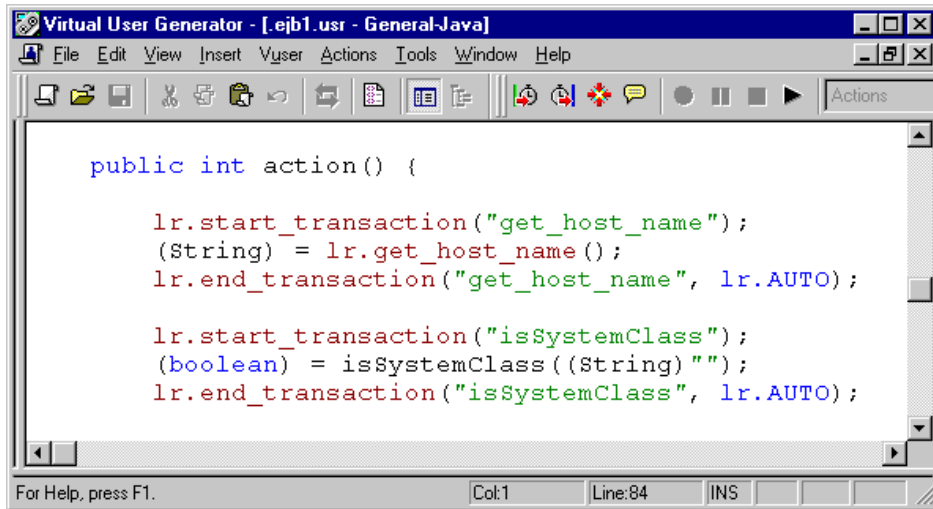
Class Name Path

The **FullClassName** option displays the complete package and class name in the Java Function navigator. This option does not affect the way the functions are added into the script—it only affects the way the classes are displayed in the navigator. By default, this option is set to false. If your packages have many classes and you are unable to view the package and class names at the same time, you should enable this option.

FullClassName enabled	FullClassName disabled
 <pre> mercury.inspect ├── mercury.inspect.Buffer ├── mercury.inspect.Client ├── mercury.inspect.CloseConnectionExceptio └── mercury.inspect.Javalnspect </pre>	 <pre> mercury.inspect ├── Buffer ├── Client ├── CloseConnectionExce └── Javalnspect </pre>

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.



The screenshot shows the Virtual User Generator (VuGen) interface with a Java script snippet. The script is as follows:

```
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 interface includes a menu bar (File, Edit, View, Insert, Vuser, Actions, Tools, Window, Help), a toolbar with various icons, and a status bar at the bottom indicating "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)"...);
```

Java Custom Filters

When testing your Java application, your goal is to determine how the server reacts to client requests. When load testing, you want to see how the server responds to a load of many users. With VuGen's Java Vuser, you create a script that emulates a client communicating with your server.

VuGen provides filter files that define hooking properties for commonly used methods. There are filter definitions for RMI, CORBA, JMS, and JACADA protocols. You can also define custom filters as described below.

When you record a method, the methods which are called from the recorded method either directly or indirectly, will not be recorded.

In order to record a method, VuGen must recognize the object upon which the method is invoked, along with the method's arguments. VuGen recognizes an object if it is returned by another recorded method provided that:

- the construction method of that object is hooked
- it is a primitive or a built-in object
- It supports a serializable interface.

You can create a custom filter to exclude unwanted methods. When recording a Java application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters for RMI, CORBA, JMS, and JACADA protocols were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your JAVA application's calls or exclude unnecessary calls. Custom JAVA protocols, proprietary enhancements and extensions to the default protocols, and data abstraction all require a custom filter definition.

Guidelines for Setting Filters

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, VuGen allows you to record with a stack trace that logs all of the methods that were called by your application. In order to record with stack trace set the log level to **Detailed**. For more information, see "Defining an Effective Filter" on page 261.

Once you determine the required methods and classes, you include them by updating the **user.hooks** file. When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Note: If you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this, is that if you rerecord a script after modifying the filters, it will overwrite all manual changes.

Determining which Elements to Include or Exclude

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- ▶ **Top Down Approach.** An approach in which you include the relevant package and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined layer which implements all client-server activity without involving any GUI elements.
- ▶ **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT packages and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- ▶ If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- ▶ If you identify a non-client/server call in your script, exclude its method in the filter.
- ▶ During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen serializes it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by deserializing it.
- ▶ VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the `lr.deserialize()` method in your script. For more information see "Using the Serialization Mechanism" on page 272.
- ▶ Exclude all activity which involves GUI elements.

- Add classes for utilities that may be required for the script to be compiled.

Defining an Effective Filter

When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

To define an effective filter:

- 1** Create a new filter based on one of the built-in filters by modifying the **user.hooks** file which is located in the product's **classes** directory.
- 2** Open the Recording Options (Ctrl+F7) and select the **Log Options** node. Select the Log Level to **Detailed**.
- 3** Record your application. Click **Start Record** (Ctrl + R) to begin and **Stop** (Ctrl + F5) to end.
- 4** View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain and correlate, you should customize the script's filter.
- 5** Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) or by viewing a Stack Trace of the script.
- 6** Set the filter to include the relevant methods. For more information, see "Determining which Elements to Include or Exclude" on page 260.
- 7** Record the application again. You should always rerecord the application after modifying the filter.
- 8** Repeat steps 4 through 7 until you get a simple script which can be easily maintained and correlated.
- 9** Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information about the built-in correlation mechanism, see Chapter 16, "Java - Correlating."

Note: Do not modify any of the other .hooks file as it might damage the VuGen recorder.

Adding custom hooks to the default recorder is a complicated task and should be considered thoroughly as it has both functional and performance consequences.

Incorrect hooking definitions can lead to incorrect scripts, slow recording, and application freeze-up.

Hooks Files Structure

The following section describes the structure of a typical hooks file:

```
[Hook-Name]
class    = MyPackage.MyClass
method  = MyMethod
signature = ()V
ignore_cl =
ignore_mtd =
ignore_tree =
cb_class = mercury.ProtocolSupport
cb_mtd =
general_cb = true
deep_mode = soft | hard
make_methods_public = true | false
lock = true | false
```

The hook files are structured as .ini files where each section represents a hook definition. Regular expressions are supported in some of the entries. Any entry that uses regular expression must start with a '!'.

Hook-Name

Specifies the name of this section in the hooks file. Hook-Name must be unique across all hooks files. A good practice is to give the fully qualified class name and method. For example:

```
[javax.jms.Queue.getQueueName]
```

Class

A fully qualified class name. Regular expression can be used to include several classes from the same package, a whole package, several packages, or any class that matches a name. For example:

```
Class = !javax\jms\.*
```

Method

The simple name of the method to include. Regular expressions can be used to include more than one method from the class. For example:

```
Method = getQueueName
```

Signature

The standard Java internal type signature of the method. To determine the signature of a method, run the command `javap -s class-name` where `class name` is the fully qualified name of the class. Regular expressions can be used to include several methods with the same name, but with different arguments. For example:

```
Signature = !.*
```

ignore_cl

A specific class to ignore from the classes that match this hook. This can be a list of comma separated class names. Each item in the list can contain a regular expression. If an item in the list contains a regular expression, prepend a '!' to the class name. For example:

```
Ignore_cl = !com.hp.jms.Queue,!com\hp\.*
```

ignore_mtd

A specific method to ignore. When the loaded class method matches this hook definition, this method will not be hooked. The method name must be the simple method name followed by the signature (as explained above). To ignore multiple methods, list them in a comma separated list. To use a regular expression, prepend a '!' to the method name. For example:

```
Ignore_cl = open, close
```

ignore_tree

A specific tree to ignore. When the name of the class matches the ignore tree expression, any class that inherits from it will not be hooked, if it matches this hooks definition. To ignore multiple trees, list them in a comma separated list. To use a regular expression, prepend a '!' to the class name. This option is relevant only for hooks that are defined as deep.

cb_class

The callback class that gets the call from the hooked method. It should always be set to **mercury.ProtocolSupport**.

cb_mtd

A method in the callback class that gets the call from the hooked method. If omitted, it uses the default, **general_rec_func**. For cases where you just need to lock the subtree of calls, use **general_func** instead.

general_cb

The general callback method. This value should always be set to **true**.

Deep_mode

Deep mode refers to classes and interfaces that inherit or implement the class or interface that the hook is listed for. The inherited classes will be hooked according to the type of hook: **Hard**, **Soft**, or **Off**.

- ▶ **Hard**. Hooks the current class and any class that inherits from it. If regular expressions exist, they are matched against every class that inherits from the class in the hook definition. Interface inheritance is treated the same as class inheritance.
- ▶ **Soft**. Hooks the current class and any class that inherits from it, only if the methods are overridden in the inheriting class. If the hook lists an interface, then if a class implements this interface those methods will be hooked. If they exist in classes that directly inherit from that class they will also be hooked. However, if the hook lists an interface and a class implements a second interface that inherits from this interface, the class will not be hooked.

Note: Regular expressions are not inherited but converted to actual methods.

- **Off.** Only the class listed in the hook definition and the direct inheriting class will be hooked. If the hook lists an interface, only classes that directly implement it will be hooked.

make_methods_public:

Any method that matches the hook definition will be converted to public. This is useful for custom hooks or for locking a sub tree of calls from a non-public method.

Note that this applies only during record. During replay, the method will use the original access flags. In the case of non-public methods, it will throw `java.lang.VerifyError`.

Lock

When set to **true**, it locks the sub tree and prevents the calling of any method originating from the original method.

When set to **false**, it will unlock the sub tree, record any method originating from the current method (if it is hooked), and invoke the callback.

16

Java - Correlating

VuGen's correlation allows you to link Java Vuser functions by using the results of one statement as input to another.

This chapter includes:

- ▶ About Correlating Java Scripts on page 268
- ▶ Standard Correlation on page 269
- ▶ Advanced Correlation on page 269
- ▶ String Correlation on page 271
- ▶ Using the Serialization Mechanism on page 272

About Correlating Java Scripts

Vuser scripts containing Java code often contain dynamic data. When you record a Java Vuser script, the dynamic data is recorded into scripts, but cannot be re-used during replay. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. In many cases, correlation will solve the problem by enabling you to use the results of one statement as input to another.

VuGen's Java recorder attempts to automatically correlate statements in the generated script. It only performs correlation on Java objects. When it encounters a Java primitive (byte, character, boolean, integer, float, double, short, and long) during recording, the argument values appear in the script without association to variables. VuGen automatically correlates all objects, arrays of objects, and arrays of primitives. Note that Java arrays and strings are also considered objects.

VuGen employs several levels of correlation: Standard, Enhanced, Strings. You enable or disable correlation from the Recording options. An additional method of Serialization can be used to handle scripts where none of the former methods can be applied. For more information, see "Using the Serialization Mechanism" on page 272.

Standard Correlation

Standard correlation refers to the automatic correlation performed during recording for simple objects, excluding object arrays, vectors, and container constructs.

When the recorded application invokes a method that returns an object, VuGen's correlation mechanism records these objects. When you run the script, VuGen compares the generated objects to the recorded objects. If the objects match, the same object is used. The following example shows two CORBA objects `my_bank` and `my_account`. The first object, `my_bank`, is invoked; the second object, `my_account`, is correlated and passed as a parameter in final line of the segment:

```
public class Actions {  
  
    // Public function: init  
    public int init() throws Throwable {  
  
        Bank my_bank = bankHelper.bind("bank", "shunra");  
        Account my_account = accountHelper.bind("account", "shunra");  
  
        my_bank.remove_account(my_account);  
    }  
    :  
}
```

Advanced Correlation

Advanced or **deep** correlation refers to the automatic correlation performed during recording for complex objects, such as object arrays and CORBA container constructs.

The deep correlation mechanism handles CORBA constructs (structures, unions, sequences, arrays, holders, 'any's) as containers. This allows it to reference inner members of containers, additional objects, or different containers. Whenever an object is invoked or passed as a parameter, it is also compared against the inner members of the containers.

In the following example, VuGen performs deep correlation by referencing an element of an array. The `remove_account` object receives an `account` object as a parameter. During recording, the correlation mechanism searches the returned array `my_accounts` and determines that its sixth element should be passed as a parameter.

```
public class Actions {

    // Public function: init
    public int init() throws Throwable {

        my_banks[] = bankHelper.bind("banks", "shunra");
        my_accounts[] = accountHelper.bind("accounts", "shunra");

        my_banks[2].remove_account(my_accounts[6]);
    }
    :
}
```

The following segment further illustrates enhanced correlation. The script invokes the `send_letter` object that received an `address` type argument. The correlation mechanism retrieves the inner member, `address`, in the sixth element of the `my_accounts` array.

```
public class Actions {

    // Public function: init
    public int init() throws Throwable {

        my_banks = bankHelper.bind("bank", "shunra");
        my_accounts = accountHelper.bind("account", "shunra");

        my_banks[2].send_letter(my_accounts[6].address);
    }
    :
}
```

String Correlation

String correlation refers to the representation of a recorded value as an actual string or a variable. When you disable string correlation (the default setting), the actual recorded value of the string is indicated explicitly within the script. When you enable string correlation, it creates a variable for each string, allowing you to use it at a later point in the script.

In the following segment, string correlation is enabled—you store the value returned from the `get_id` method in a string type variable for use later on in the script.

```
public class Actions {  
  
    // Public function: init  
    public int init() throws Throwable {  
  
        my_bank = bankHelper.bind("bank", "shunra");  
        my_account1 = accountHelper.bind("account1", "shunra");  
        my_account2 = accountHelper.bind("account2", "shunra");  
  
        string = my_account1.get_id();  
        string2 = my_account2.get_id();  
        my_bank.transfer_money(string, string2);  
    }  
:  
}
```

You set the correlation method from the **Correlation** tab in the recording options.

- ▶ **Correlate Strings.** Correlate strings in script during recording. If you disable this option, the actual recorded values are included in the script between quotation marks. If this option is disabled, all other correlation options are ignored (disabled by default).
- ▶ **Correlate String Arrays.** Correlate strings within string arrays during recording. If you disable this option, strings within arrays are not correlated and the actual values are placed in the script (enabled by default).

- ▶ **Advanced Correlation.** Enables correlation on complex objects such as arrays and CORBA container constructs and arrays. This type of correlation is also known as deep correlation (enabled by default).
- ▶ **Correlation Level.** Determines the level of deep correlation—how many inner containers to search.
- ▶ **Correlate Collection Type.** Correlate objects contained in a Collection class for JDK 1.2 or higher (disabled by default).

Using the Serialization Mechanism

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

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

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user directory. Additional parameters are saved as new binary data files, numbered sequentially. VuGen generates the following code:

```
public class Actions {

    // Public function: init
    public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)jr.deserialize(0, false);
        map.set_point(p);
    }
    :
}
```


The integer passed to **lr.deserialize** represents the number of binary data files in the Vuser directory.

To parameterize the recorded value, use the public `setLocation` method (for information, see the JDK function reference). The following example uses the `setLocation` method to set the value of the object, `p`.

```
public class Actions {
    // Public function: init
    public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
        p.setLocation(2,9);
        map.set_point(p);
    }
    :
    :
}
```

In certain instances the public method of **setLocation** is not applicable. As an alternative, you can use the API of your class that incorporate get or set accessor methods. If you are working with AUT classes that do not have get/set methods or use private methods, or if you are unfamiliar with the classes' API, you can use VuGen's built-in serialization mechanism. This mechanism allows you to expand objects in their ASCII representation and manually parameterize the script. You enable this mechanism in the Recording Options dialog box (see Chapter 18, "Java Recording Options" in *Volume I-Using VuGen* for more information).

VuGen generates an **lr.deserialize** method that deserializes the data or displays complex data structures as serial strings. Once the structure is broken down to its components, it is easier to parameterize. The **lr.deserialize** method receives two arguments, a string and an integer. The string is the parameter's value that is to be substituted during replay. The integer is the index number of binary file to load.

If you choose not to expand objects in your script by clearing the Unfold Serialized Objects check box, you can control the serialization mechanism by passing arguments to the `lr.deserialize` method. The first argument is an integer indicating the number of binary files to load. The second integer is a boolean value:

- true** Use VuGen's serialization mechanism.
- false** Use the standard Java serialization mechanism.

The following segment shows a generated script in which the serialization mechanism was enabled.

```
public class Actions {
    // Public function: init
    public int init() throws Throwable {
        _string = "java.awt.Point __CURRENT_OBJECT = {" +
            "int x = "#5#" +
            "int y = "#8#" +
            "}";
        java.awt.Point p = (java.awt.Point)lr.deserialize(_string,0);
        map.set_point(p);
    }
    :
}
```

The string values are placed between delimiters. The default delimiter is "#". You can change the delimiter in the **Serialization** tab of the recording options. Delimiters are used to speed up the parsing of the string during replay.

When modifying the string, you must maintain the following rules:

- ▶ Order of lines may not be changed. The parser reads the values one-by-one—not the member names.
- ▶ Only values between two delimiters may be modified.
- ▶ Object references may not be modified. Object references are indicated only to maintain internal consistency.

- "_NULL_" can appear as a value, representing the Java null constant. You can replace it with string type values only.
- Objects may be deserialized anywhere in the script. For example, you can deserialize all objects in the **init** method and use the values in the **action** method.
- Maintain internal consistency for the objects. For example, if a member of a vector is **element count** and you add an element, you must modify the element count.

In the following segment, a vector contains two elements:

```
public class Actions {
    // Public function: init
    public int init() throws Throwable {
        _string = "java.util.Vector CURRENTOBJECT = {" +
            "int capacityIncrement = "#0#" +
            "int elementCount = #2#" +
            "java/lang/Object elementData[] = {" +
            "elementData[0] = #First Element#" +
            "elementData[1] = #Second Element#" +
            "elementData[2] = _NULL_" +
            ....
            "elementData[9] = _NULL_" +
            "}" +
        "};
        _vector = (java.util.Vector)lr.deserialize(_string,0);
        map.set_vector(_vector);
    }
    :
}
```

In the following example, one of the vector's elements was changed—a "_NULL_" value was changed to "Third element". In coordination with the addition of the new element, the "elementCount" member was modified to "3".

```
public class Actions {
    // Public function: init
    public int init() throws Throwable {
        _string = "java.util.Vector CURRENTOBJECT = {" +
            "int capacityIncrement = "#0#" +
            "int elementCount = #3#" +
            "java/lang/Object elementData[] = {" +
                "elementData[0] = #First Element#" +
                "elementData[1] = #Second Element#" +
                "elementData[2] = #Third Element#" +
                ....
                "elementData[9] = _NULL_" +
            "}" +
        "};";
        _vector = (java.util.Vector)lr.deserialize(_string,0);
        map.set_vector(_vector);
    }
    :
}
```

Due to the complexity of the serialization mechanism, which opens up the objects to ASCII representation, opening large objects while recording may increase the time required for script generation. To decrease this time, you can specify flags which will improve the performance of the serialization mechanism.

When adding **lr.deserialize** to your script, we recommend that you add it to the **init** method—not the **action** method. This will improve performance since VuGen will only deserialize the strings once. If it appears in the **action** method, VuGen will deserialize strings for every iteration.

The following list shows the available options which you set in **Serialization** tab of the recording options:

- Serialization Delimiter
- Unfold Serialized Objects
- Unfold Arrays
- Limit Array Entries
- Ignore Serialized Objects

See Chapter 18, "Java Recording Options" in *Volume I-Using VuGen* for complete information on the recording options.

17

Enterprise Java Beans (EJB) Protocol

VuGen helps you create a script for testing Enterprise Java Beans (EJB) objects on your application server.

This chapter includes:

- ▶ About EJB Testing on page 279
- ▶ Working with the EJB Detector on page 280
- ▶ Creating an EJB Testing Vuser on page 285
- ▶ Understanding EJB Vuser Scripts on page 289
- ▶ Running EJB Vuser Scripts on page 295

About EJB Testing

VuGen provides several tools for developing a script that tests Java applications. For generating a Vuser script through recording, use the Java Record and Replay Vuser. For creating a script through programming, use the Custom Java Vuser type.

EJB Testing Vusers differ from the standard Java Vusers in that VuGen automatically creates a script to test or tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. VuGen's EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to test or tune, and VuGen generates a script that emulates each of the EJB's methods.

It creates transactions for each method so that you can measure its performance and locate problems. In addition, each method is wrapped in a **try and catch** block for exception handling.

Note that in order to create EJB testing scripts, the EJB Detector must be installed and active on the application server host. The Detector is described in the following sections.

VuGen also has a built-in utility for inserting methods into your script. Using this utility, you display all of the available packages, select the desired methods, and insert them into your script. For more information, see "Running EJB Vuser Scripts" on page 295.

Working with the EJB Detector

The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

Installing the EJB Detector

The EJB Detector can be installed and invoked on the application server's machine or alternatively, on the client machine. To run the EJB Detector on the client machine you must have a mounted drive to the application server machine.

To install the EJB detector agent:

- 1 Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
- 2 Unzip the `<LR_root>\ejbcomponent\ejbdetector.jar` file into the EJB Detector directory.

Running the EJB Detector

The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

The EJB Detector can run from the command-line, or from a batch file.

To run the EJB Detector from the command line:

- 1** Before running the EJB Detector from the command line, add the DETECTOR_HOME\classes and the DETECTOR_HOME\classes\xerces.jar to the CLASSPATH environment variable.
- 2** If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the CLASSPATH:

For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar

For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar

- 3** If your EJBs use additional classes directory or .jar files, add them to the CLASSPATH.

4 To run the EJB Detector from the command-line, use the following string:

```
java EJBDetector [search root dir] [listen port]
```

search root dir	<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>

To run the EJB Detector from a batch file:

You can launch the EJB detector using a batch file, **EJB_Detector.cmd**. This file resides in the root directory of the EJB Detector installation, after you unzip **ejbdetector.jar**.

1 Open **env.cmd** in the EJB Detector root directory, and modify the following variables according to your environment:

JAVA_HOME	the root directory of JDK installation
DETECTOR_INS_DIR	the root directory of the Detector installation

APP_SERVER_DRIVE	the drive hosting the application server installation
APP_SERVER_ROOT	Follow these guidelines: BEA WebLogic Servers 4.x and 5.x. Specify the application server root directory. BEA WebLogic Servers 6.x. Specify full path of the domain folder. WebSphere Servers 3.x. Specify the full path of the deployed EJBs folder. WebSphere Servers 4.0. Specify the application server root directory. Oracle OC4J. Specify the application server root directory. Sun J2EE Server. Specify the full path to the deployable .ear file or directory containing a number of .ear files.
EJB_DIR_LIST (optional)	list of directories/files, separated by ';' and containing deployable .ear/.jar files, and any additional classes directory or .jar files or used by your EJBs under test.

2 Save **env.cmd**.

3 If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested, as well as the following vendor EJB classes, to the CLASSPATH in the env file:

- For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar
- For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar

4 Run the **EJB_Detector.cmd** or **EJB_Detector.sh** (Unix platforms) batch file to collect information about the deployable applications containing EJBs, for example:

```
C:\>EJB_Detector [listen_port]
```

where **listen_port** is an optional argument specifying a port number on which the EJB Detector will listen for incoming requests (default is 2001).

EJB Detector Output and Log Files

You can examine the output of the EJB Detector to see if it has detected all the active EJBs. The output log shows the paths being checked for EJBs. At the end of the scan, it displays a list of the EJBs that were found, their names and locations. For example:

```

Checking EJB Entry: f:/weblogic/myserver/ejb_basic_beanManaged.jar...
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_statefulSession.jar...
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_statelessSession.jar...
----- Found 3 EJBs -----
** PATH: f:/weblogic/myserver/ejb_basic_beanManaged.jar
- BEAN: examples.ejb.basic.beanManaged.AccountBean
** PATH: f:/weblogic/myserver/ejb_basic_statefulSession.jar
- BEAN: examples.ejb.basic.statefulSession.TraderBean
** PATH: f:/weblogic/myserver/ejb_basic_statelessSession.jar
- BEAN: examples.ejb.basic.statelessSession.TraderBean

```

If no EJBs were detected (that is, "Found 0 EJBs"), check that the EJB jar files are listed in the "Checking EJB Entry:..." lines. If they are not listed, check that the **search root dir** path is correct. If they are being inspected but still no EJBs are detected, check that these EJB jar files are deployable (can be successfully deployed into an application server). A deployable jar file should contain the Home Interface, Remote Interface, Bean implementation, the Deployment Descriptor files (xml files, or .ser files), and additional vendor-specific files.

If you still encounter problems, set the debug properties in the **detector.properties** file, located in the DETECTOR_HOME\classes directory, to retrieve additional debug information.

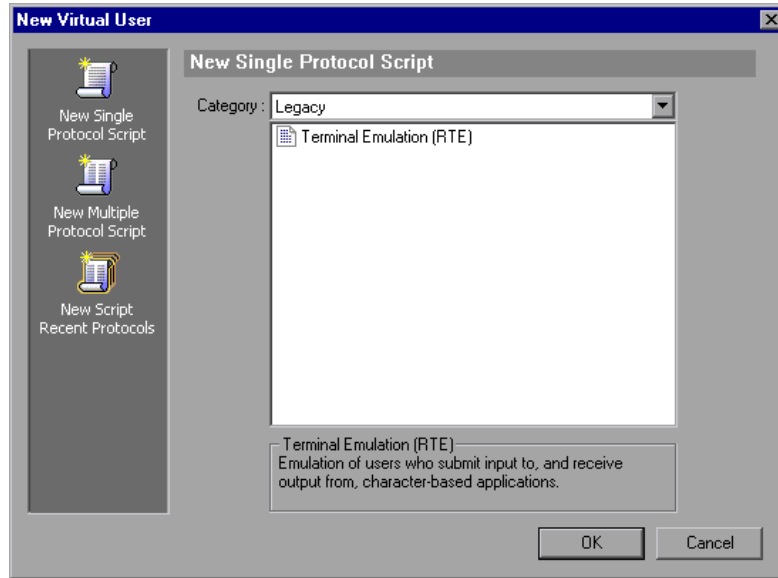
After the EJBs are detected, the HTTP Server is initialized and waits for requests from the VuGen EJB-Testing Vuser. If there are problems in this communication process, enable the property **webserver.enableLog** in the **webserver.properties** file located in the DETECTOR_HOME\classes directory.

This enables printouts of additional debug information, and other potentially important error messages in the **webserver.log** file.

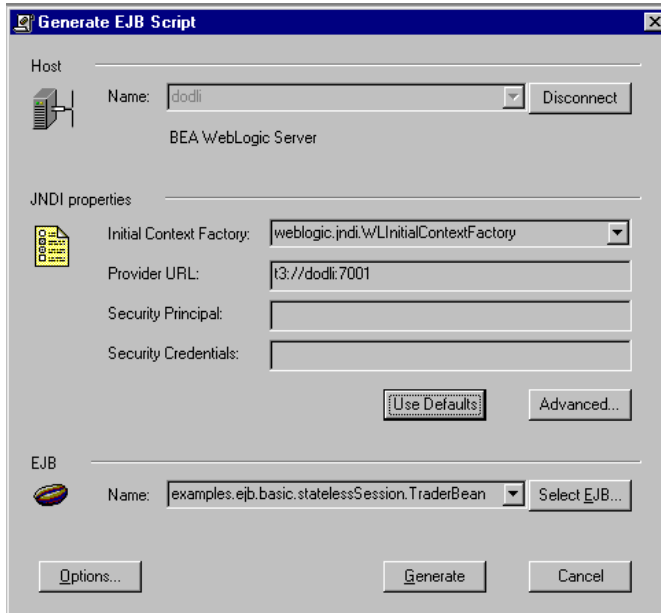
Creating an EJB Testing Vuser

To create an EJB Vuser script:

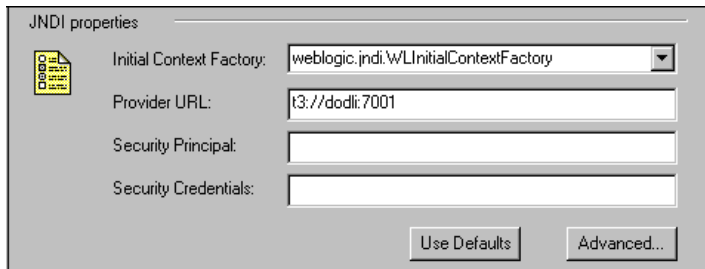
- 1 Select **File > New** or click the **New** button. The New Virtual User dialog box opens.



- 2 Select **EJB Testing** from the **Enterprise Java Beans** category and click **OK**. VuGen opens a blank Java Vuser script and opens the Generate EJB Script dialog box.



- 3 Specify a machine on which VuGen's EJB Detector is installed. Note that the Detector must be running in order to connect. Click **Connect**. The **JNDI properties** section is enabled.



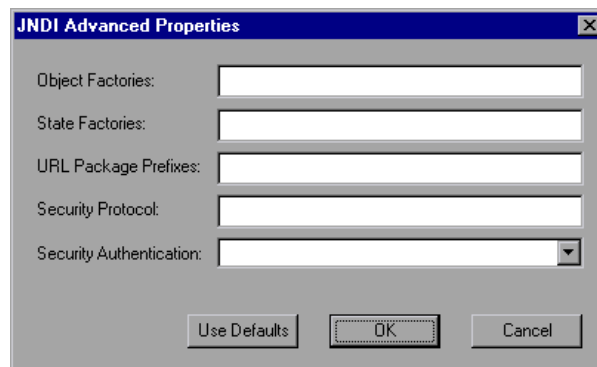
- 4** The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory** and the **Provider URL**.

If your application server requires authentication, enter the user name in the **Security Principal** box and a password in the **Security Credentials** box.

Here are the default values of the two JNDI mandatory properties:

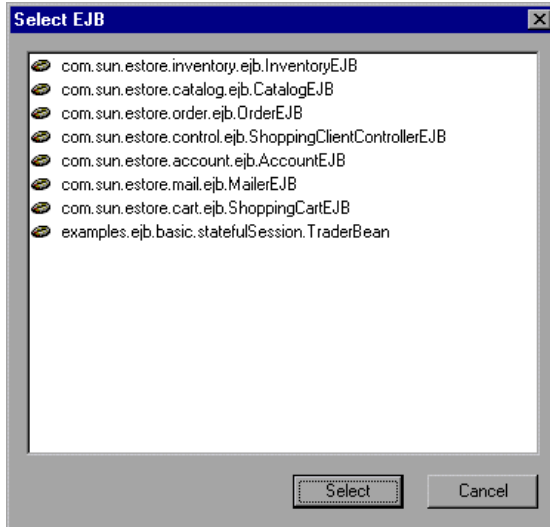
Type	Initial Context Factory	Provider URL
WebLogic	weblogic.jndi.WLInitialContextFactory	t3://<appserver_host>:7001
WebSphere 3.x	com.ibm.ejs.ns.jndi.CNInitialContextFactory	iiop://<appserver_host>:900
WebSphere 4.x	com.ibm.websphere.naming.WsnInitialContextFactory	iiop://<appserver_host>:900
Sun J2EE	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.

Understanding EJB Vuser Scripts

VuGen generates a script that tests your EJB, based on the JNDI (Java Naming and Directory Interface) properties you specified when creating the Vuser script. JNDI is Sun's programming interface used for connecting Java programs to naming and directory services such as DNS and LDAP.

Each EJB Vuser script contains three primary parts:

- Locating the EJB Home Using JNDI
- Creating an Instance
- Invoking the EJB Methods

Locating the EJB Home Using JNDI

The first section of the script contains the code that retrieves the JNDI properties. Using the specified context factory and provider URL, it connects to the application server, looks up the specified EJB and locates the EJB Home.

In the following example, the JNDI Context Factory is `weblogic.jndi.WLInitialContextFactory`, the URL of the provider is `t3://dod:7001` and the JNDI name of the selected EJB is `carmel.CarmelHome`.

```
public class Actions
{
    public int init() {
        CarmelHome _carmelhome = null;
        try {
            // get the JNDI Initial Context
            java.util.Properties p = new java.util.Properties();
            p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
            p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001");
            javax.naming.InitialContext _context = new javax.naming.InitialContext(p);

            // lookup Home Interface in the JNDI context and narrow it
            Object homeobj = _context.lookup("carmel.CarmelHome");
            _carmelhome =
(CarmelHome)javax.rmi.PortableRemoteObject.narrow(homeobj, CarmelHome.class);

        } catch (javax.naming.NamingException e) {
            e.printStackTrace();
        }
    }
}
```

Note: If the script is generated with an EJB Detector running on the client rather than an application server, you must manually modify the URL of the provider. For example, in the following line, the provider specifies `dod` as the EJB detector host name:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001")
```

Replace the recorded host name with the application server name, for example:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://bealogic:7001")
```

You can specify the provider URL before recording, so you don't have to modify it manually, in the **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.

Running EJB Vuser Scripts

After you generate a script for your EJB testing, and make the necessary modifications, you are ready to run your script. The EJB script allows you to perform two types of testing: functional and load. The functional testing verifies that the EJB, functions properly within your environment. The load testing allows you to evaluate the performance of the EJB when executing many users at one time.

To run a functional test:

- 1** Modify the default values that were automatically generated.
- 2** Insert value checks using the `lr.value_check` method. These methods were generated as comments in the script (see "Invoking the EJB Methods" on page 293).
- 3** Insert additional methods, and modify their default values. For more information, see the section on inserting Java functions in Chapter 14, "Java Protocols - Recording."
- 4** Set the general run-time settings for the script. For more information, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.
- 5** Set the Java VM run-time settings: Specify all additional classpaths and additional VM parameters. Make sure to include your application server EJB classes. The actual classes of the EJB under test are saved in the Vuser directory and retrieved automatically during replay. For information about specifying additional classpaths and setting the Java VM run-time settings, see Chapter 24, "Java and EJB Run-Time Settings" in *Volume I-Using VuGen*.
- 6** For **Websphere 3.x** users:
 - Using the IBM JDK 1.2 or higher:
 - Add the `<WS>\lib\ujc.jar` to the classpath.
 - Using the Sun JDK 1.2.x:
 - Remove the file `<JDK>\jre\lib\ext\iiimp.jar`
 - Copy the following files from the `<WS>\jdk\jre\lib\ext` folder to the `<JDK>\jre\lib\ext` directory: `iiprt.jar`, `rmiorb.jar`.

- ▶ Copy the 'ujc.jar' from the <WS>\lib folder, to <JDK>\jre\lib\ext.
- ▶ Copy the file <WS>\jdk\jre\bin\ioser12.dll to the <JDK>\jre\bin folder.

where <WS> is the home folder of the WebSphere installation and <JDK> is the home folder of the JDK installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

7 For **WebSphere 4.0** users:

Make sure that you have valid Java environment on your machine of IBM JDK1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<WS>/lib/webshpere.jar;  
<WS>/lib/j2ee.jar;
```

Where <WS> is the home directory of the WebSphere installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

Note: If your application server is installed on a UNIX machine or if you are using WebSphere 3.0.x, you must install IBM JDK 1.2.x on the client machine to obtain the required files.

8 For **Oracle OC4J** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher (JDK1.3 preferable). Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<OC4J>/orion.jar;<OC4J>/ejb.jar;<OC4J>/jndi.jar; ;<OC4J>/xalan.jar;  
<OC4J>/crimson.jar
```


where <OC4J> is the home folder of the application server installation.

9 For **Sun J2EE** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<J2EE>/j2ee.jar;<AppClientJar>
```

where <J2EE> is the home folder of the application server installation and <AppClientJar> is the full path to the application client jar file created automatically by the sdk tools during the deployment process.

10 For **WebLogic 4.x - 5.x** Users:

Make sure that you have valid Java environment on your machine (path & classpath). Open the Run-Time Settings dialog box and select the Java VM node. Add the following two entries to the Additional Classpath section:

```
<WL>/classes;<WL>/lib/weblogicaux.jar
```

where <WL> is the home folder of the WebLogic installation.

11 For **WebLogic 6.x** and **7.0** users:

Make sure that you have valid Java environment on your machine (path & classpath). WebLogic 6.1 requires JDK 1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entry to the Additional Classpath section:

```
<WL>/lib/weblogic.jar; // Weblogic 6.x  
<WL>/server/lib/weblogic.jar // Weblogic 7.x
```

where <WL> is the home folder of the WebLogic installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

- 12** Run the script. Click the **Run** button or select **Vuser > Run**. View the Execution Log node to view any run-time errors. The execution log is stored in the **mdrv.log** file in the script's folder. A Java compiler (Sun's javac), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

18

Citrix Protocol

VuGen allows you to record the actions of a Citrix client communicating with its server using the Citrix ICA protocol. The resulting script is called a Citrix Vuser script.

The optional **Citrix Agent** helps you create an intuitive script that provides built-in synchronization. For more information, see Chapter 19, "Citrix - Agent for Citrix Presentation Server." See "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 318 for valuable tips on creating scripts.

This chapter includes:

- About Creating Citrix Vuser Scripts on page 300
- Getting Started with Citrix Vuser Scripts on page 301
- Setting Up the Client and Server on page 302
- Recording Tips on page 304
- Setting the Citrix Display Settings on page 306
- Viewing and Modifying Citrix Vuser Scripts on page 307
- Synchronizing Replay on page 308
- Understanding ICA Files on page 316
- Using Citrix Functions on page 317
- Tips for Replaying and Troubleshooting Citrix Vuser Scripts on page 318

About Creating Citrix Vuser Scripts

Citrix Vuser scripts emulate the Citrix ICA protocol communication between a Citrix client and server. VuGen records all activity during the communication and creates a Vuser script.

When you perform actions on the remote server, VuGen generates functions that describe these actions. Each function begins with a **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. (See "Recording with VuGen" in *Volume I-Using VuGen*.) 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**).

You can view and edit the recorded script from VuGen's main window. The API calls that were recorded during the session are displayed in VuGen, allowing you to track your actions.

Getting Started with Citrix Vuser Scripts

This section provides an overview of the process of developing Citrix ICA Vuser scripts using VuGen. In addition, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 318.

To develop a Citrix ICA script:

1 Make sure that your client and server are configured properly.

For general information about these settings, see "Setting Up the Client and Server" on page 302.

2 Record the actions using VuGen.

Invoke VuGen and create a new Vuser script. Insert bitmap and text synchronization during recording as described in "Synchronizing Replay" on page 308.

For general information about recording, see "Recording with VuGen" in *Volume I-Using VuGen*.

3 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Configure the Citrix display options.

Configure the display options for replaying Citrix Vusers. These options let you show the Citrix client during replay and open a snapshot when an error occurs. For details, see "Setting the Citrix Display Settings" on page 306.

6 Configure the Run-Time settings.

The Run-Time settings control Vuser behavior during script execution. These settings include pacing, logging, think time, and connection information.

For details about the Citrix specific Run-Time settings, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*. For information about general Run-Time settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

7 Save and run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly. While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script.

For details about running the Vuser script as a standalone test, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 318 and "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Setting Up the Client and Server

Before creating a script, make sure you have a supported Citrix client installed on your machine, and that your server is properly configured. This section describes:

- Client Versions
- Server Configuration

Client Versions

In order to run your script, you must install a Citrix client on each Load Generator machine. If you do not have a client installed, you can download one from the Citrix Website www.citrix.com under the **download** section.

VuGen supports all Citrix clients with the exception of versions 8.00, version 6.30.1060 and earlier, and Citrix Web clients.

Server Configuration

To record in VuGen, you need to configure the Citrix server in the following areas:

Installing the MetaFrame Sever

Make sure the MetaFrame server (3, 4, or 4.5) is installed. To check the version of the server, select **Citrix Connection Configuration** on the server's console toolbar and select **Help > About**.

Configuring the MetaFrame Server to Close Sessions

Configure the Citrix server to completely close a session. After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time that client opens a new connection. Consequently, a new connection by the same client will face the same workspace from which it disconnected previously. It is preferable to allow each new test run to use a clean workspace.

The make sure that you have a clean workspace for each test, you must configure the Citrix server not to save the previous session. Instead, it should reset the connection by disconnecting from the client each time the client times-out or breaks the connection.

To configure the server:

- 1** Open the Citrix Connection Configuration dialog box. Select **Programs > Citrix > Administration Tools > Citrix Connection Configuration Tool**.
- 2** Select the ica-tcp connection name and select **Connection > Edit**. Alternatively, double-click on the connection. The Edit Connection dialog box opens.
- 3** Click the **Advanced** button. The Advanced Connection Settings dialog box opens.
- 4** In the bottom section of the dialog box, clear the **inherit user config** check box adjacent to the **On a broken or timed-out connection** list box. Change the entry for this list box to **reset**.

5 Click OK.

Recording Tips

When recording a script, be sure to follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. If you plan to record a simple Citrix ICA session, use a single protocol script. When recording an NFUSE Web Access session, however, you must create a multi-protocol script for Citrix ICA and Web(HTML/HTTP), to enable the recording of both protocols. For more information, see "Recording with VuGen" in *Volume I-Using VuGen*.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Recording with VuGen" in *Volume I-Using VuGen*.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start > Programs > Microsoft Word**, be sure to click on the word **Programs**.

Do not Resize Windows

Although VuGen supports the resizing of windows during recording the session, we recommend that you do not move or resize them while recording. To change the size or position of a window, double-click on the relevant **Sync on Window** step in the script's Tree view and modify the window's coordinates.

Make Sure Resolution Settings are Consistent

To insure successful bitmap synchronization, make sure that the resolution settings match. On the recording machine, check the settings of the ICA client, the Recording Options, and the Run Time settings. On the load generators, check the settings of the ICA client, and make sure that they are consistent between all load generators and recording machines. If there is an inconsistency between the resolutions, the server traffic increases in order to make the necessary adjustments.

Add Manual Synchronization Points

While waiting for an event during recording, such as the opening of an application, we recommend that you add manual synchronization points, such as **Sync on Bitmap** or **Sync on Text**. For details, see "Synchronizing Replay" on page 308.

Disable Client Updates

Disable client updates when prompted by the Citrix client. This will prevent forward compatibility issues between VuGen and newer Citrix clients that were not yet tested.

Windows Style

For **Sync on Bitmap** steps, record windows in the "classic" windows style—not the XP style.

To change the Windows style to "classic":

- 1 Click in the desktop area.
- 2 Select **Properties** from the right-click menu.
- 3 Select the Theme tab.
- 4 Select **Windows Classic** from the Theme drop down list.

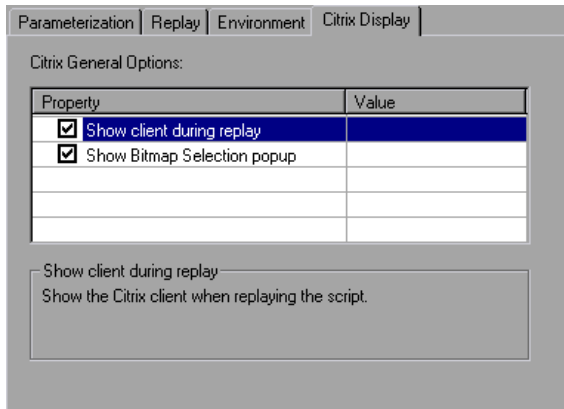
- 5 Click **OK**.

Setting the Citrix Display Settings

Before running your Citrix Vuser script, you can set several display options to be used during replay. Although these options increase the load upon the server, they are useful for debugging and analyzing your session.

To set the Citrix display options:

- 1 Open the General Options dialog box. Select **Tools > General Options** in the main VuGen window.
- 2 Select the **Citrix Display** tab.



- 3 Select **Show client during replay** to display the Citrix client when replaying the Vuser script.
- 4 Select **Show Bitmap Selection popup** to issue a popup message when you begin to work interactively within a snapshot. VuGen issues this message when you select the right-click menu option **Insert Sync Bitmap** or **Insert Get Text**, before you select the bitmap or text.
- 5 Click **OK**.

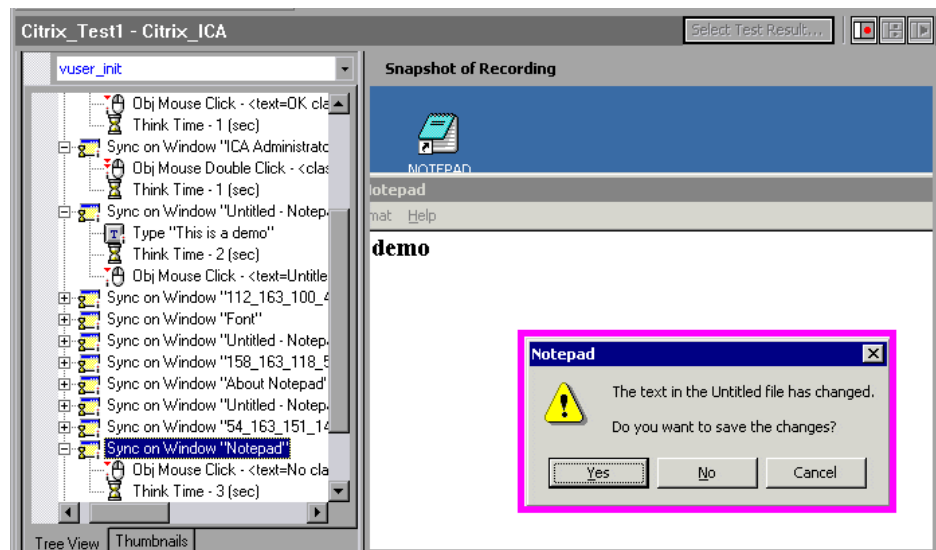
Viewing and Modifying Citrix Vuser Scripts

You can view the contents of your Vuser script in VuGen's Script view or Tree view. For general information about viewing a script, see "Introducing Service TestVuGen" in *Volume I-Using VuGen*.

In Tree view, you can view a Citrix Vuser's snapshots. Each step has an associated snapshot. In addition to displaying the client window, the snapshot also highlights the object upon which the action was performed.

- ▶ For the **Mouse** steps, a small pink square indicates where the user clicked.
- ▶ For **Sync on Bitmap**, a pink box encloses the bitmap area.

For **Sync on Window**, a pink box encloses the entire window. In the following example, the snapshot shows the **Sync On Window** step. Notepad's confirmation box is enclosed by a box indicating the exact window on which the operation was performed.



Note that VuGen saves snapshots as bitmap files in the script's data/snapshots directory. You can determine the name of the snapshot file by checking the function's arguments.

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

After recording, you can manually add steps to the script in either Script view or Tree view. For information about the various script views, see "Introducing Service TestVuGen" in *Volume I-Using VuGen*.

In addition to manually adding new functions, you can add new steps interactively for Citrix Vusers, directly from the snapshot. Using the right-click menu, you can add bitmap synchronizations. When the Citrix Agent is installed on the Citrix server machine, you can also add text and object synchronizations from the right-click menu. For more information, see Chapter 19, "Citrix - Agent for Citrix Presentation Server."

To insert a function interactively:

- 1** Click on a step within Tree view. Make sure that a snapshot is visible.
- 2** Right-click and select one of the commands. A dialog box opens with the available properties.
- 3** Modify the desired properties and click **OK**. VuGen inserts the step into your script.

Synchronizing Replay

When running a script, it is often necessary to synchronize the actions to insure a successful replay. Synchronization refers to the timing of events within your script, waiting for windows and objects to become available before executing an action. For example, you may want to check whether a certain window has opened before attempting to press a button within the window.

VuGen automatically generates functions that synchronize the actions during replay. In addition, you can add manual synchronization functions.

Automatic Synchronization

During recording, VuGen automatically generates steps that help synchronize the Vuser's replay of the script:

- Sync on Window
- Sync on Obj Info
- Sync on Text

Sync on Window

The **Sync On Window** step instructs the Vuser to wait for a specific event before resuming replay. The available events are **Create** or **Active**. The Create event waits until the window is created. The Active event waits until the window is created and then activated (in focus). Usually VuGen generates a function with a CREATE event. If, however, the next instruction is a keyboard event, VuGen generates a function with an ACTIVE event.

In Script view, the corresponding function call to the **Sync On Window** step is `ctrx_sync_on_window`.

Sync on Obj Info

The **Sync On Obj Info** step instructs the Vuser to wait for a specific object property before resuming replay. The available attributes are **Enabled**, **Visible**, **Focused**, **Text**, **Checked**, **Lines**, or **Item**. The Enabled, Visible, Focused, and Checked attributes are boolean values that can receive the values **true** or **false**. The other attributes require a textual or numerical object value.

A primary objective of this step is to wait for an object to be in focus before performing an action upon it.

VuGen automatically generates Sync On Obj Info steps when the Citrix agent is installed and the Use Citrix Agent Input in Code Generation option is enabled in the Recording options. By default, this Recording option is enabled. For more information, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

```
ctrx_sync_on_obj_info("Run=snapshot9", 120, 144, TEXT, "OK",
                    CTRX_LAST);
```

Sync on Text

The Text Synchronization step, **Sync On Text**, instructs the Vuser to wait for a text string to appear at the specified position before continuing. When replaying **Sync On Text**, Vusers search for the text in the rectangle whose modifiable coordinates are specified in the step's properties.

With an agent installation (see Chapter 19, "Citrix - Agent for Citrix Presentation Server"), you can instruct VuGen to automatically generate a text synchronization step before each mouse click or double-click. By default, automatic text synchronization is disabled. For information on how to enable this Recording option, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

Note, that even if you record a script with the option disabled, if you enable the option and regenerate the script, VuGen will insert text synchronization calls throughout the entire script. For more information, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

You can manually add text synchronizations for individual steps both during and after recording as described in "Manual Synchronization" on page 311.

In Script view, the corresponding function call to the **Sync On Text** step is **ctrx_sync_on_text_ex**.

The following segment shows a `ctrx_sync_on_text_ex` function that was recorded during a Citrix recording with the HP Citrix Agent installed and text synchronization enabled.

```
ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,224,
"snapshot1", CTRX_LAST);
ctrx_sync_on_text_ex (196, 198, 44, 14, "OK", "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);
ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198, LEFT_BUTTON, 0, "ICA
Seamless Host Agent=snapshot2", CTRX_LAST);
```

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

Manual Synchronization

In addition to the automatic synchronization, you can manually add synchronization both during and after recording. A common use of this capability is where the actual window did not change, but an object within the window changed. Since the window did not change, VuGen did not detect or record a **Sync on Window**.

For example, if you want the replay to wait for a specific graphic image in a browser window, you insert manual synchronization. Or, if you are recording a large window with several tabs, you can insert a synchronization step to wait for the new tab's content to open.

The following section describes synchronizing on a bitmap. For information on adding a **Sync on Text** manually, see "Text Retrieval" on page 329.

Manually Adding Synchronization During Recording

To add synchronization during recording, you use the floating toolbar. The **Sync On Bitmap** function lets you to mark an area within the client window that needs to be in focus before resuming replay.



Sync on Bitmap

To mark a bitmap area for synchronization:



- 1 Click the **Insert Sync on Bitmap** button on the toolbar.
- 2 Mark a rectangle around the desired bitmap. In Tree view, VuGen generates a Sync on Bitmap step after the current step. In Script view, VuGen generates a `ctx_sync_on_bitmap` function with the selected coordinates as arguments.

```
ctx_sync_on_bitmap(93, 227, 78, 52,
                  "66de3122a58baade89e63698d1c0d5dfa", CTRX_LAST);
```

During replay, Vusers look for the bitmap at the specified coordinates, and wait until it is available before resuming the test.

Manually Adding Synchronization After Recording

You can also add synchronization after the recording session. To add a synchronization step, right-click in the snapshot window and select a synchronization option:

- ▶ **Sync on Bitmap.** Waits until a bitmap appears
- ▶ **Sync on Obj Info.** Waits until an object's attributes have the specified values (agent installations only)
- ▶ **Sync on Text.** Waits until the specified text is displayed (agent installations only)

During recording, the bitmaps generated for the **Sync on Bitmap** step are saved under the script's **data/snapshots** directory. If synchronization fails during replay, VuGen generates a new bitmap that you can examine to determine why synchronization failed. VuGen displays both bitmaps in the Failed Bitmap Synchronization dialog box. For more information, see "Failed Bitmap Synchronization" on page 315.

The bitmap name has the format of `sync_bitmap_<hash_value>.bmp`. It is stored in the script's output directory, or for a scenario or profile, wherever the output files are written.

Additional Synchronizations

In addition, you can add several other steps that affect the synchronization indirectly:

- ▶ Setting the Waiting Time
- ▶ Checking if a Window Exists or Closed
- ▶ Waiting for a Bitmap Change

Setting the Waiting Time

The **Set Waiting Time** step sets a waiting time for the other Citrix synchronization functions. This setting applies to all functions that follow it within the script. For example, if your **Sync on Window** steps are timing out, you can increase the default timeout of 60 seconds to 180.

To insert this step, select **Insert > Add Step > Set Waiting Time**.

Checking if a Window Exists or Closed

The Win Exist step checks if a window is visible in the Citrix client. By adding control flow statements, you can use this function to check for a window that does not always open, such as a warning dialog box. In the following example, `ctrx_win_exist` checks whether a browser was launched. The second argument indicates how long to wait for the browser window to open. If it did not open in the specified time, it double-clicks its icon.

```
if (!ctrx_win_exist("Welcome",6, CTRX_LAST))
    ctrx_mouse_double_click(34, 325, LEFT_BUTTON, 0, CTRX_LAST)
```

To insert this step, select **Insert > Add Step > Win Exist**.

Another useful application for this step is to check if a window has been closed. If you need to wait for a window to close, you should use a synchronization step such as **UnSet Window** or `ctrx_unset_window`.

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

Waiting for a Bitmap Change

In certain cases, you do not know what data or image will be displayed in an area, but you do expect it to change. To emulate this, you can use the **Sync on Bitmap Change** step or its corresponding function, `ctrx_sync_on_bitmap_change`. Perform a right-click in the snapshot, and select an **Insert Sync on Bitmap** from the right-click menu. VuGen inserts the step or function at the location of the cursor.

The syntax of the functions is as follows:

```
ctrx_sync_on_bitmap (x_start, y_start, width, height, hash, CTRX_LAST);
ctrx_sync_on_bitmap_change (x_start, y_start, width, height,
    [initial_wait_time,] [timeout,]
    [initial_bitmap_value,] CTRX_LAST);
```

The following optional arguments are available for **ctrx_sync_on_bitmap_change**:

- ▶ initial wait time value—when to begin checking for a change.
- ▶ a timeout—the amount of time in seconds to wait for a change to occur before failing.
- ▶ initial bitmap value—the initial hash value of the bitmap. Vusers wait until the hash value is different from the specified initial bitmap value.

In the following example, the recorded function was modified and assigned an initial waiting time of 300 seconds and a timeout of 400 seconds.

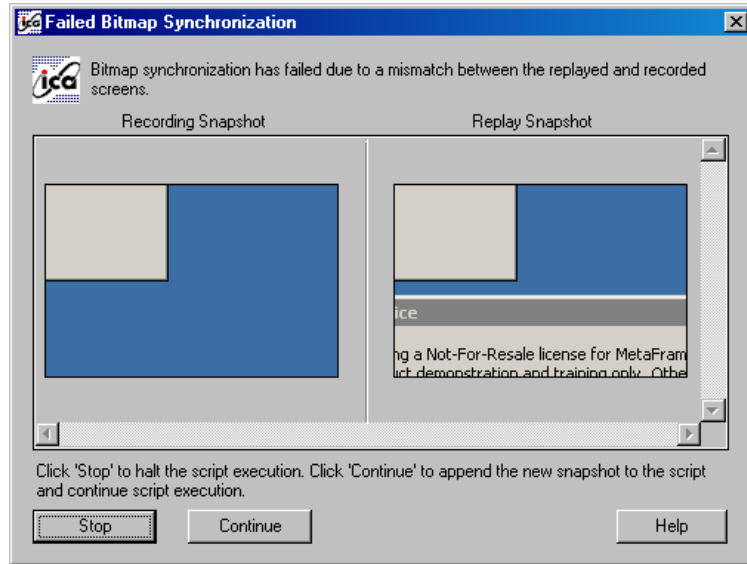
```
ctrx_sync_on_bitmap_change(93, 227, 78, 52,  
300,400, "66de3122a58baade89e63698d1c0d5dfa",CTRX_LAST);
```

Note: If you are using **Sync on Bitmap**, make sure that the Configuration settings in the Controller, Load Generator machine, and screen are the same. Otherwise, VuGen may be unable to find the correct bitmaps during replay. For information on how to configure the client settings, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

Failed Bitmap Synchronization

The Failed Bitmap Synchronization dialog box opens when there is a mismatch between the Recording and Replay snapshots during script replay.

You can indicate whether or not you want to mark the mismatch as an error or adopt the changes.



When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution. You can specify Continue on Error for a specific function as described in "Continuing on Error" on page 321.
- **Continue.** Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.

Understanding ICA Files

Citrix ICA client files are text files that contain configuration information for the applications accessed through the Citrix client. These files must have an .ica extension and must conform to the following format:

```
[WFClient]
Version=
TcpBrowserAddress=

[ApplicationServers]
AppName1=

[AppName1]
Address=
InitialProgram=#
ClientAudio=
AudioBandwidthLimit=
Compress=
DesiredHRES=
DesiredVRES=
DesiredColor=
TransportDriver=
WinStationDriver=

Username=
Domain=
ClearPassword=
```

Note: When you load an ICA file using the Recording Options, VuGen saves the file together with your script, eliminating the need to copy the ICA file to each load generator machine.

The following example shows a sample ICA file for using Microsoft Word on a remote machine through the Citrix client:

```
[WFClient]
Version=2
TcpBrowserAddress=235.119.93.56

[ApplicationServers]
Word=

[Word]
Address=Word
InitialProgram=#Word
ClientAudio=On
AudioBandwidthLimit=2
Compress=On
DesiredHRES=800
DesiredVRES=600
DesiredColor=2
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

Username=test
Domain=user_lab
ClearPassword=test
```

For more information, see the Citrix Website www.citrix.com.

Using Citrix Functions

During a Citrix recording session, VuGen generates functions that emulate the communication between a client and a remote server. The generated functions have a **ctx** prefix. You can also manually edit any of the functions into your Vuser script after the recording session. For example, **ctx_obj_mouse_click** emulates a mouse click for a specific object.

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.

Tips for Replaying and Troubleshooting Citrix Vuser Scripts

The following sections provide guidelines and tips for Citrix Vusers in the following areas:

- ▶ Replay Tips
- ▶ Debugging Tips

For recording tips, see "Recording Tips" on page 304.

Replay Tips

Wildcards

You can use wildcards (*) in defining window names. This is especially useful where the window name may change during replay, by its suffix or prefix.

In the following example, the title of the **Microsoft Internet Explorer** window was modified with a wildcard.

```
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0,  
"Welcome to MSN.com - Microsoft Internet Explorer");  
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0,  
"* - Microsoft Internet Explorer");
```

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

Set Initialization Quota

To prevent overloading by multiple Vusers while connecting, set an initialization quota of 4 to 10 Vusers (depending on the capacity of the server) or apply ramp-up initialization using the Scheduler.

Enable Think Time

For best results, do not disable think time in the Run-Time settings. Think time is especially relevant before the `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 "Regenerating a Vuser Script" in *Volume I-Using VuGen*.

Set Consistency Between Machines

If you intend to replay the script on another machine, make sure that the following items are consistent between the record and replay machines: Window Size (resolution), Window Colors, System Font and the other Default Options settings for the Citrix client. These settings affect the hash value of bitmaps, and inconsistencies may cause replay to fail. To view the Citrix Client settings, select an item from the Citrix program group and select **Application Set Settings** or **Custom Connection Settings** from the right-click menu. Select the Default Options tab.

Increasing the Number of Vusers per Load Generator Machine

Load Generator machines running Citrix Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine, also known as the GDI (Graphics Device Interface). To increase the number of Vusers per machine, you can open a terminal server session on the machine which acts as an additional load generator.

The GDI count is Operating System dependent. The actual GDI (Graphics Device Interface) count for a heavily loaded machine using LoadRunner is approximately 7,500. The maximum available GDI on Windows 2000 machines is 16,384.

For more information on creating a terminal server session, see the Terminal Services topics in the HP LoadRunner Controller.

Note: By default, sessions on a terminal server use a 256-color set. If you intend to use a terminal session for load testing, make sure to record on machines with a 256-color set.

Debugging Tips

Single Client Installation

If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine. To verify that only one client is installed, open the Add/Remove Programs dialog box from the Control Panel and make sure that there is only one entry for the Citrix ICA client.

Add Breakpoints

Add breakpoints to your script in VuGen to help you determine the problematic lines of code.

Synchronize Your Script

If replay fails, you may need to insert synchronization functions into your script to allow more time for the desired windows to come into focus. Although you can manually add a delay using `lr_think_time`, we recommend that you use one of the synchronization functions discussed in "Synchronizing Replay" on page 308.

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 `ctx_sync_on_bitmap` function are saved under the script's **data** directory. The bitmap name has the format of `hash_value.bmp`. If synchronization fails during replay, the generated bitmap is written to the script's output directory, or if you are running it in a scenario, to wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.

Show Vusers

To show Vusers during a scenario, enter the following in the Vuser command line box: `-lr_citrix_vuser_view`. In the Controller, open the Vuser Details dialog box and click **More** to expand the dialog box. Note that this will affect the scalability of the test, so this should only be done to examine a problematic Vuser's behavior.

To reduce the effect on the script's scalability, you can show the details for an individual Vuser by adding the Vuser's ID at the end of the command line: `-lr_citrix_vuser_view <VuserID>`.

To open multiple Vusers, place a comma-separated list of IDs after the command line. Do not use spaces, but you may use commas or dashes. For example, `1,3-5,7` would show Vusers 1,3,4,5, and 7, but would not show Vuser 2, 6 or any Vuser with an ID higher than 7.

View Recording and Replay Logs

To see detailed information about the recording, view the Recording and Replay logs in the Output window. To open the Output window, select **View > Output Window**.

To view the Recording Log, select the **Recording Log** tab. VuGen displays a detailed log of all functions that were generated by the recording and the warning messages and errors that were issued during that time.

```
[Citrix Recorder (874: 8a4)] ctrx_set_connect_opt("APPLICATION", "#DeskTop");
[Citrix Recorder (874: 8a4)] ctrx_set_connect_opt("NETWORK_PROTOCOL", "TCP/IP + H
[Citrix Recorder (874: 8a4)] ctrx_connect_server("plato", "test" "*****" "qalab
[Citrix Recorder (874: 8a4)] window size = 800x600 , session color depth = 4 bits
[Citrix Recorder (874: 8a4)] ctrx_create_window "Starting DeskTop" flag=1 style=-
[Citrix Recorder (874: 8a4)] Foreground window = 131114
[Citrix Recorder (874: 8a4)] ctrx_active_window("Starting DeskTop");
[Citrix Recorder (874: 8a4)] Foreground window = 65562
[Citrix Recorder (874: 8a4)] ctrx_create_window "Winlogon generic control dialog"
[Citrix Recorder (874: 8a4)] ctrx_delete_window "Winlogon generic control dialog"
[Citrix Recorder (874: 8a4)] ctrx_create_window "Citrix License Warning Notice" f
[Citrix Recorder (874: 8a4)] ctrx_create_window "Citrix License Warning Notice_2"
[Citrix Recorder (874: 8a4)] ctrx_create_window "Starting DeskTop_2" flag=0 style
[Citrix Recorder (874: 8a4)] ctrx_active_window("Starting DeskTop_2");
```

To view the Replay Log, select the **Replay Log** tab. VuGen provides a description of all actions performed by VuGen and any warnings and errors that were issued during the replay.

```
Action.c(10): Waiting for logon
Action.c(10): Logon succeeded
Action.c(12): Waiting for window "Citrix License Warning Notice_2" to exist
Action.c(12): Ensuring window "Citrix License Warning Notice_2" is at (155, 232, 49
Action.c(14): Waiting for window "ICA Seamless Host Agent" to exist
Action.c(14): Start notification arrived from the Agent, Agent version = "8.1.0.454
Action.c(14): WARNING: The Citrix Agent is not the same version as the Citrix Recor
You are advised to install the same version of the software on both client and
Action.c(14): Ensuring window "ICA Seamless Host Agent" is at (0, 0, 391, 224)
Action.c(16): Waiting for "TEXT" information for object at (232, 154) within window
Action.c(18): Executing left mouse click at (232, 154) in window "Citrix License Wa
Action.c(20): Waiting for "TEXT" information for object at (191, 196) within window
```

To go directly to the step in the script associated with the log message, double-click on the message in the Replay log.

Note that the extent of information in the Replay log depends on the Log Run-Time settings. For more information, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

19

Citrix - Agent for Citrix Presentation Server

The Agent for Citrix Presentation Server is an optional utility that you can install on the Citrix server. It provides you with several important benefits that can enhance your script:

This chapter includes:

- About the Agent for Citrix Presentation Server on page 325
- Using the Agent for Citrix Presentation Server Features on page 326
- Data Execution Prevention (DEP) and Citrix Performance on page 330
- Installing the Citrix Presentation Server Agent on page 332
- Effects and Memory Requirements of the Citrix Agent on page 333
- Sample Script on page 333

About the Agent for Citrix Presentation Server

The Agent for Citrix Presentation Server is an optional utility that you can install on the Citrix server. It provides enhancements to the normal Citrix functionality. It is provided in the product's installation disk and you can install it on any Citrix server.

The agent provides you with the following benefits:

- Intuitive and readable scripts
- Built-in synchronization
- Detailed Information about relevant objects

- Ability to work interactively within the Client window

Using the Agent for Citrix Presentation Server Features

The Agent for Citrix Presentation Server provides enhancements to the normal Citrix functionality. The following sections describe these enhancements and provide details and sample scripts.

Object Detail Recording

When the Agent for Citrix Presentation Server is installed, VuGen records specific information about the active object instead of general information about the action. For example, VuGen generates **Obj Mouse Click** and **Obj Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows the same mouse-click action recorded with and without the agent installation. Note that with an agent, VuGen generates `ctrx_obj_XXX` functions for all of the mouse actions, such as click, double-click and release.

```
/* Without Agent Installation */
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0, test3.txt - Notepad);

/* With Agent Installation */
ctrx_obj_mouse_click("<text=test3.txt - Notepad class=Notepad>" 573,
    61, LEFT_BUTTON, 0, test3.txt - Notepad=snapshot21, CTRX_LAST);
```

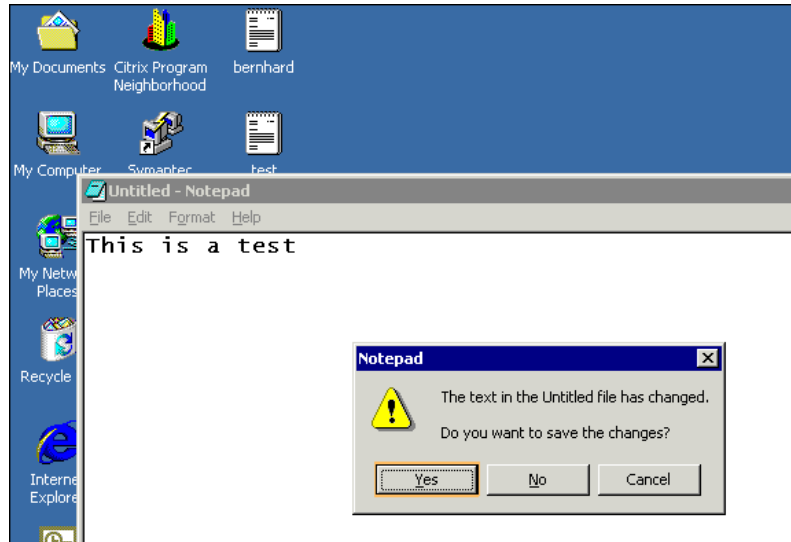
In the example above, the first argument of the `ctrx_obj_mouse_click` function contains the text of the window's title and the class, Notepad. Note that although the agent provides additional information about each object, Vusers only access objects by their window name and its coordinates.

Active Object Recognition

The agent installation lets you see which objects in the client window are detected by VuGen. This includes all Windows Basic Objects such as edit boxes, buttons, and item lists in the current window.

To see which objects were detected, you move your mouse through the snapshot. VuGen highlights the borders of the detected objects as the mouse passes over them.

In the following example, the **Yes** button is one of the detected objects.



Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When no agent is installed, you are limited to the **Insert Mouse Click**, **Insert Mouse Double Click**, **Insert Sync on Bitmap** and **Insert Get Bitmap Value**. If you are using a 256-color set, the **Insert Sync on Bitmap** and **Get Bitmap Value** steps are not available from the right-click menu.

When the Agent for Citrix Presentation Server is installed, the following additional options are available from the right-click menu of window in focus:

- ▶ **Obj Get Info** and **Sync on Obj Info**. Provide information about the state of the object: ENABLED, FOCUSED, VISIBLE, TEXT, CHECKED, and LINES.

- ▶ **Insert Sync on Obj Info.** Instructs VuGen to wait for a certain state before continuing. This is generated as a `ctrx_sync_on_obj_info` function.
- ▶ **Insert Obj Get Info.** Retrieves the current state of any object property. This is generated as a `ctrx_get_obj_info` function.
- ▶ **Insert Sync on Text and Get Text.** These are discussed in the section "Text Retrieval" on page 329.

These commands are interactive—when you insert them into the script, you mark the object or text area in the snapshot.

In the following example, the `ctrx_sync_on_obj_info` function provides synchronization by waiting for the Font dialog box to come into focus.

```
ctrx_sync_on_obj_info("Font", 31, 59, FOCUSED, "TRUE", CTRX_LAST);
```

Utilizing VuGen's ability to detect objects, you can perform actions on specific objects interactively, from within the snapshot.

To insert a function interactively using the agent capabilities:

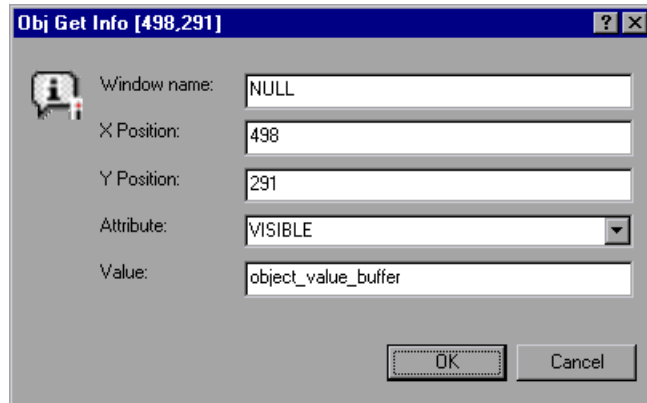
- 1** Click at a point within the tree view to insert the new step. Make sure that a snapshot is visible.
- 2** Click within the snapshot.
- 3** To mark a bitmap, right-click on it and select **Insert Sync on Bitmap**.

VuGen issues a message indicating that you need to mark the desired area by dragging the cursor. Click **OK** and drag the cursor diagonally across the bitmap that you want to select.

When you release the mouse, VuGen inserts the step into the script after the currently selected step.

- For all other steps, move your mouse over snapshot objects to determine which items are active—VuGen highlights the borders of active objects as the mouse passes over them.

Right-click and select one of the Insert commands. A dialog box opens with the step's properties.



Set the desired properties and click **OK**. VuGen inserts the step into your script.

Text Retrieval

With the agent installed, VuGen lets you save standard text to a buffer. Note that VuGen can only save true text—not a graphical representation of text in the form of an image.

You save the text using the **Sync On Text** step either during or after recording.

To retrieve a text string:

- During recording: Click the **Insert Sync On Text** button on the toolbar.

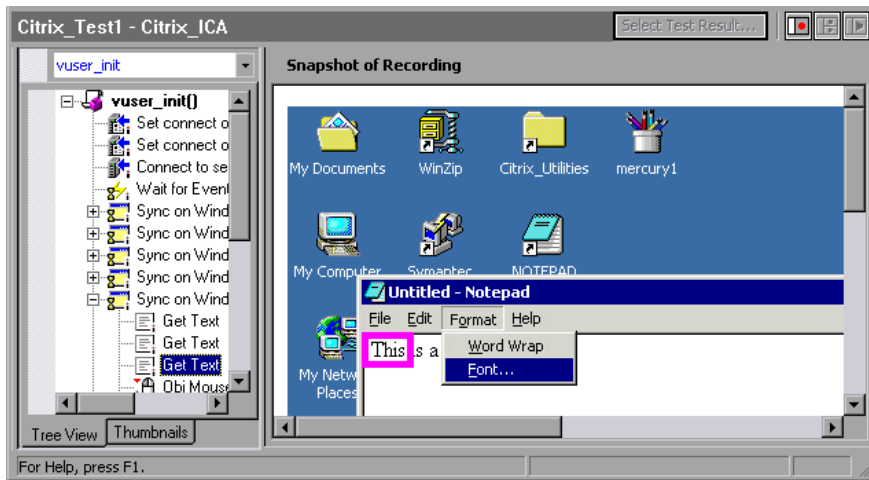


After recording: Select **Insert Sync On Text** from the snapshot's right-click menu. The Bitmap Selection dialog opens, indicating that you are inserting a synchronization or informational function and that you need to mark an area.

- 2 Click at the corner of the text that you want to capture, drag the mouse diagonally to mark the text you want to save, and release the mouse button.
- 3 If you add the step during recording, VuGen places a **Insert Sync On Text** step at the current location and saves the text to a buffer.

If you are adding the step after recording, VuGen prompts you with the Sync On Text Ex dialog box, allowing you to manually specify text.

VuGen marks the saved text with a pink box. In the following snapshot, the **Insert Sync On Text** step retrieved the text **This**.



Data Execution Prevention (DEP) and Citrix Performance

DEP is a security feature included in Windows version XP Service Pack 2 and later. DEP can interfere with some functions of the Agent for Citrix Presentation Server. We recommend that the users disable DEP to ensure that the agent runs properly.

To disable DEP

- 1 From the desktop, right click **My Computer** and select **Properties**. Alternatively, click **Start > Control Panel > System**. The **System Properties** dialog box opens.

- 2** Click the **Advanced** tab.
- 3** In the **Startup and Recovery** pane, click **Settings**.
- 4** Under **System startup**, click **Edit**.
- 5** Locate the `/noexecute` element and change the value to **AlwaysOff**.
- 6** Save the file and reboot your machine.

Installing the Citrix Presentation Server Agent

The installation file for the Agent for Citrix Presentation Server is located on the LoadRunner installation disk, under the **Additional Components\Agent for Citrix Presentation Server** folder.

Note that the agent should only be installed on your Citrix server machine—not Load Generator machines.

If you are upgrading the agent, make sure to uninstall the previous version before installing the next one (see uninstallation instructions below).

To install the Agent for Citrix Presentation Server:

- 1 If your server requires administrator permissions to install software, log in as an administrator to the server.
- 2 If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

- 3 Locate the installation file, **Setup.exe**, on the product installation disk in the **Additional Components\Agent for Citrix Presentation Server\Win32 or Win64** directory.
- 4 Follow the installation wizard to completion.

Note: After installation the agent will only be active for LoadRunner invoked Citrix sessions—it will not be active for users who start a Citrix session without LoadRunner.

To disable the agent, you must uninstall it.

To uninstall the Agent for Citrix Presentation Server:

- 1 If your server requires administrator privileges to remove software, log in as an administrator to the server.

- 2 Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Citrix Presentation Server 32 or 64** and click **Change/Remove**.

Effects and Memory Requirements of the Citrix Agent

When you run Citrix Vusers with the agent installed, each Vuser runs its own process of **ctrxagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine (about 7%).

The memory requirements per Citrix ICA Vuser (each Vuser runs its own **ctrxagent.exe** process) is approximately 4.35 MB. To run 25 Vusers, you would need 110 MBs of memory.

Sample Script

The following script illustrates a true Citrix ICA session with an agent.

```
vuser_init ()
{
  ctrx_set_connect_opt (NETWORK_PROTOCOL, "TCP/IP + HTTP");
  ctrx_connect_server ("Plato", "test", lr_decrypt("428c4445a14409b9"), "QALab");
  ctrx_wait_for_event ("LOGON");
  ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,224,
"snapshot1", CTRX_LAST);
  ctrx_sync_on_text (196, 198, "OK", TEXT, "ICA Seamless Host Agent=snapshot2",
CTRX_LAST);
  ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198, LEFT_BUTTON, 0, "ICA
Seamless Host Agent=snapshot2", CTRX_LAST);
  lr_think_time (73);
  return 0;
}
```


20

Remote Desktop Protocol (RDP)

VuGen allows you to record the actions of a client communicating with its server using the Microsoft Remote Desktop Protocol (RDP). The resulting script is called an RDP Vuser script.

This chapter includes:

- About Microsoft Remote Desktop Protocol (RDP) Vuser Scripts on page 336
- Recording Tips on page 336
- Recording an RDP Vuser Script on page 337
- Running RDP Vuser Scripts on page 339
- Working with Clipboard Data on page 339
- Synchronizing Replay on page 342

About Microsoft Remote Desktop Protocol (RDP) Vuser Scripts

The Microsoft Remote Desktop Protocol (RDP) allows users to connect to a remote computer. For example, you can use RDP to connect to a central and powerful server for working on specific business applications or graphic terminals. This provides the user with the same look and feel as if they are working on a standalone PC.

Note: RDP versions 5.1 and later have an **Experience** tab that allows you to set various options. This tab is not supported by VuGen recording. All options are set to the ON position.

Recording Tips

When recording a script, be sure to follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. For example, to record both RDP traffic and Web responses, create a multi-protocol script for RDP and Web to enable the recording of both protocols. For more information, see "Recording with VuGen" in *Volume I-Using VuGen*.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see Chapter 5, "Recording with VuGen" in *Volume I-Using VuGen*.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

You should also configure your terminal server to end disconnected sessions. Select **Administrative Tools > Terminal Services Configuration > Connection Properties > Sessions > Override User Settings** and set the server to end disconnected sessions.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start > Programs > Microsoft Word**, be sure to click on the word **Programs**.

Recording an RDP Vuser Script

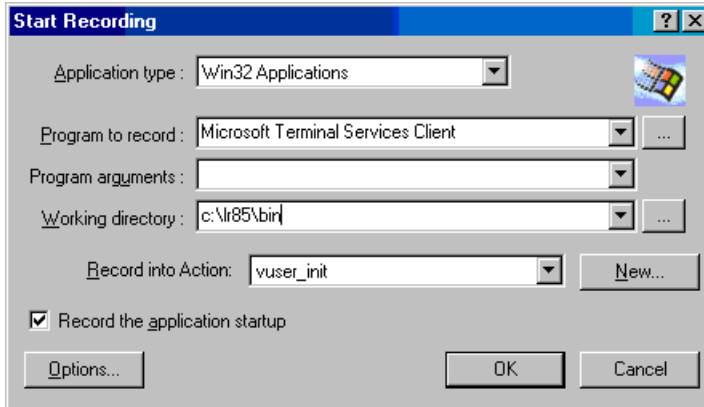
Before recording, you set the desired recording options as described above.

Note: Whenever recording an RDP script, make sure that you record the act of logging off the server. This makes sure that when you run the script you will begin with a new session.

At any point during recording, you can mark an area of the remote desktop to synchronize upon. The image is stored on disk as **snapshot_XXX.png** where **XXX** is a sequential index that starts at 1 and is increased by 1 for each image and includes images from other steps, for example, mouse clicks. A matching function appears in the script at this point during code generation.

To create an RDP Vuser script:

- 1 Select **File > New** or click the **New** button. The New Virtual User dialog box opens.
- 2 Select **Microsoft Remote Desktop Control Protocol (RDP)**. The Start Recording dialog box opens.



- 3 Click **Options** to set the Recording Options. For more information, see "RDP Recording Options" on page 287 in *Volume I-Using VuGen*.
- 4 Select the **RDP: Login** node. Select one of the session options: **Run Client**, **Connection File**, or **Default Connection File**.
- 5 Select the **RDP: Code Generation** node and enable the desired options.
- 6 During recording to select a screen region for synchronization, click the **Sync on image button** on the recording toolbar and indicate an area for synchronization.



Note: You can also add an image synchronization after recording the script. Right-click on the image snapshot and select **Insert Sync On Image** from the menu.

- 7 Stop recording and save the script.

Running RDP Vuser Scripts

Before running an RDP script, you can set the run-time settings to customize the behavior of the script. You can set general run-time settings for all protocols, such as think time, iterations, pacing, and logging, as described in "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

For RDP-specific settings, see above.

To run an RDP script:



- 1** Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the toolbar, or select **Vuser > Run-Time Settings**.
- 2** Select the **Configuration** node. Select the desired settings.
- 3** Select the **Synchronization** node. Select the desired settings.
- 4** Click **OK** to accept the run-time settings and close the dialog box.
- 5** Click the Run button or select **Vuser > Run**.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Working with Clipboard Data

VuGen allows you to copy and paste the textual contents of a clipboard during an RDP session. You can copy them locally and paste them remotely, or vice versa—copy them from the remote machine and paste them locally. The copying of text is supported in TEXT, LOCALE, and UNICODE formats.

VuGen generates separate functions when providing or saving the clipboard data.

The following example illustrates a copy operation on a local machine and a paste on a remote machine:

```
//Notifies the Remote Desktop that new data is available in the Local machine's
//clipboard. The data can be provided in three formats: TEXT, UNICODE and LOCALE
rdp_notify_new_clipboard_data(
  "StepDescription=Send local clipboard formats 1",
  "Snapshot=snapshot1.inf",
  "FormatsList=RDP_CF_TEXT|RDP_CF_UNICODE|RDP_CF_LOCALE",
  RDP_LAST );

rdp_key(
  "StepDescription=Key Press 2",
  "Snapshot=snapshot_9.inf",
  "KeyValue=V",
  "KeyModifier=CONTROL_KEY",
  RDP_LAST );

//Provides clipboard data to the Remote Desktop when it requests the data.
rdp_send_clipboard_data(
  "StepDescription=Set Remote Desktop clipboard 1",
  "Snapshot=snapshot1.inf",
  "Timeout=20",
  REQUEST_RESPONSE, "Format=RDP_CF_UNICODE", "Text=text for clipboard",
  RDP_LAST);
```

This example illustrates a copy operation on a remote machine and a paste on a local machine:

```
rdp_key(
  "StepDescription=Key Press 2",
  "Snapshot=snapshot_9.inf",
  "KeyValue=C",
  "KeyModifier=CONTROL_KEY",
  RDP_LAST);

// The function requests the Remote Desktop UNICODE text and saves it to a
//parameter
rdp_receive_clipboard_data(
  "StepDescription=Get Remote Desktop clipboard 1",
  "Snapshot=snapshot1.inf",
  "ClipboardDataFormat=RDP_CF_UNICODE",
  "ParamToSaveData=MyParam",
  RDP_LAST);
```

Normally, the Remote Desktop clipboard data is saved in UNICODE format. If the Remote Desktop requests data in the TEXT or LOCALE formats, the **rdp_send_clipboard_data** function automatically converts the content of MyParam from UNICODE into the requested format and sends it to the Remote Desktop. The Replay log indicates this conversions with an informational message. If the conversion is not possible, the step fails.

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

Correlating Clipboard Parameters

During a recording session, if the client sends the server the same data as it received, then VuGen replaces the sent data with a parameter during code generation. VuGen only performs this correlation when the received and sent data formats are consistent with one another.

The following example shows how the same parameter, **MyParam**, is used for both receiving and sending the data.

```
// Receive the data from the server
rdp_receive_clipboard_data("StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot_9.inf",
"Timeout=0",
"ClipboardDataFormat=RDP_CF_UNICODETEXT",
"ParamToSaveData=MyParam",
RDP_LAST);
...
// Send the data to the server
rdp_send_clipboard_data("StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot_9.inf",
"Timeout=10",
REQUEST_RESPONSE, "Format=RDP_CF_UNICODETEXT", "Text={MyParam}",
RDP_LAST);
```

Synchronizing Replay

An RDP session executes remotely. All keyboard and mouse handling is done on the server, and it is the server that reacts to them. For example, when you double-click an application on the desktop, it is the server that realizes a double-click took place and that the application must be loaded and displayed.

When an RDP client connects to a server, it does two things:

- ▶ It sends the server coordinates of actions. For example, 'clicked the left mouse button at coordinates (100, 100) on the screen'.
- ▶ It receives images from the server showing the current status of the screen after the action took place

The RDP client (and therefore, LoadRunner) does not know that the screen contains windows, buttons, icons, or other objects. It only knows the screen contains an image and at what coordinates the user performed the action. To allow the server to correctly interpret the actions, you set synchronization points within the script. These points instruct the script to wait until the screen on the server matches the stored screen before continuing.

To add image synchronization points to a script:

- 1** View the script in Tree view. Select **View > Tree view**.
- 2** Select an operation to which you would like to add a synchronization point.
- 3** Right-click on the image snapshot and select **Insert Synch On Image** from the menu. The cursor will change to a cross-hair.
- 4** Mark the area on the screen that you would like to synchronize upon by clicking on the left button and dragging the box to enclose the area. When you release the mouse button, the Sync on Image dialog box opens.
- 5** Click **OK**. VuGen adds a new Sync on Image step before the selected step. When you select this step, VuGen displays a snapshot that contains a pink box around the area you selected for synchronization.

The next time you replay the script, it will wait until the image returned by the server matches the image you selected.

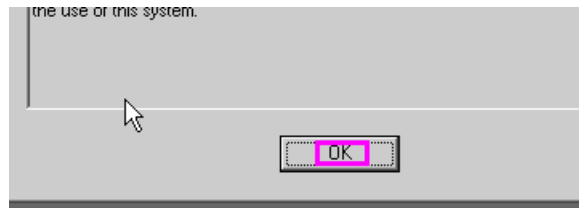
Image Synchronization Tips

Use the following guidelines for effective image synchronization:

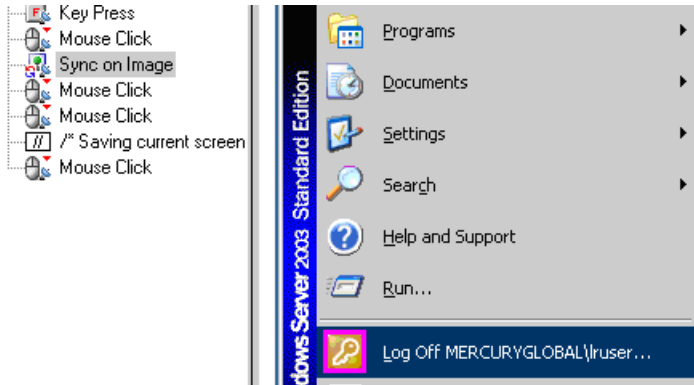
Synchronize on Smallest Significant Area

When synchronizing on an image, try to synchronize only the part of the image that is necessary. Additional details within the image may not be reproduced during replay and could result in a synchronization failure.

For example, when synchronizing on an image of a button, select only the text itself and not the dotted lines around the text as they may not appear during replay.



When synchronizing a highlighted area, try to capture only the part of the image that is not effected by the highlighting. In the following example, perform a synchronization on the Log Off icon, but not the entire button, since the highlighting may not appear during replay, and the color could vary with different color schemes.



Synchronize Before Every User Action

You need to synchronize before every mouse operation. It is also recommended that you synchronize before the first `rdp_key` / `rdp_type` operation that follows a mouse operation.

Failed Image Synchronization

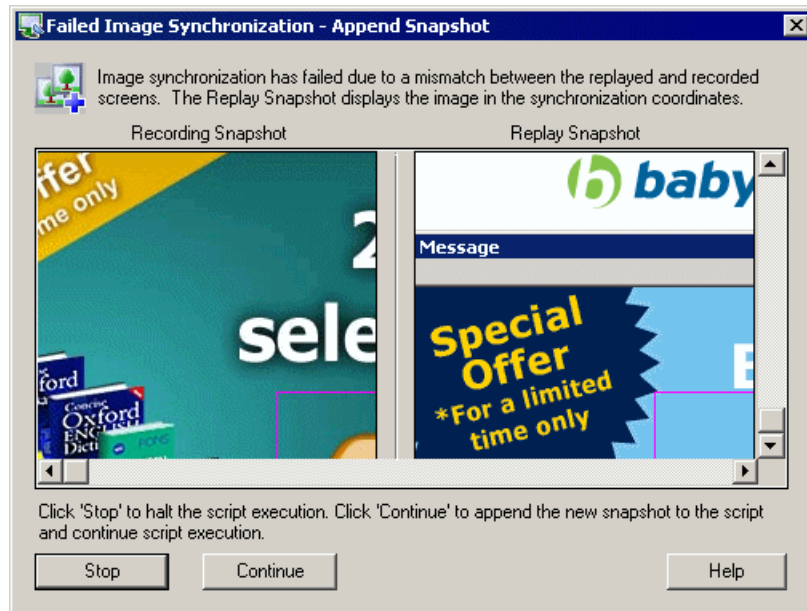
The Failed Image Synchronization dialog boxes open when there is a mismatch between the Recording and Replay snapshots during script replay. There are several Failed Image Synchronization dialog boxes:

- ▶ Append Snapshot
- ▶ Raise Tolerance
- ▶ Lower Tolerance
- ▶ No Snapshot

Note: Raising or lowering the tolerance level from the dialog box only changes the level for that step. To change the tolerance level for the whole script, change the **Default Image Sync Tolerance** setting as described in Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

Append Snapshot

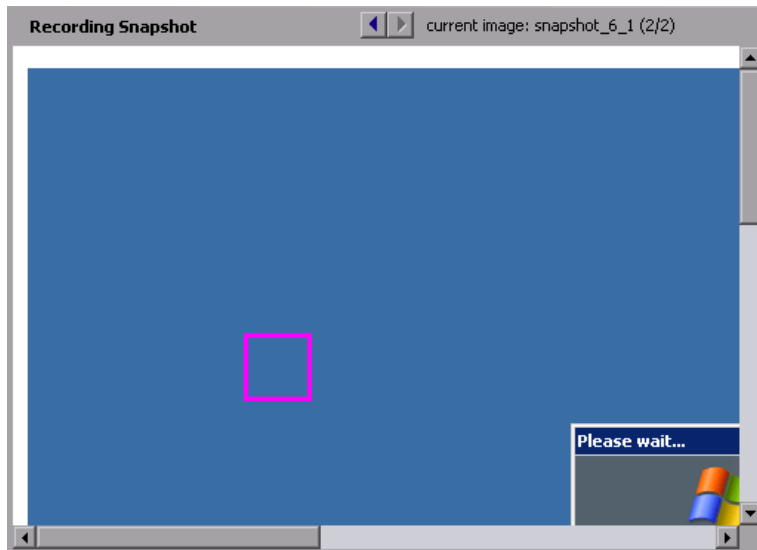
The Failed Image Synchronization - Append Snapshot dialog box opens when the replay image is so different from the recorded image that changing the tolerance level will not help. You can indicate whether or not you want to mark the mismatch as an error or adopt the changes.



When this dialog box opens, click on one of the following buttons to proceed:

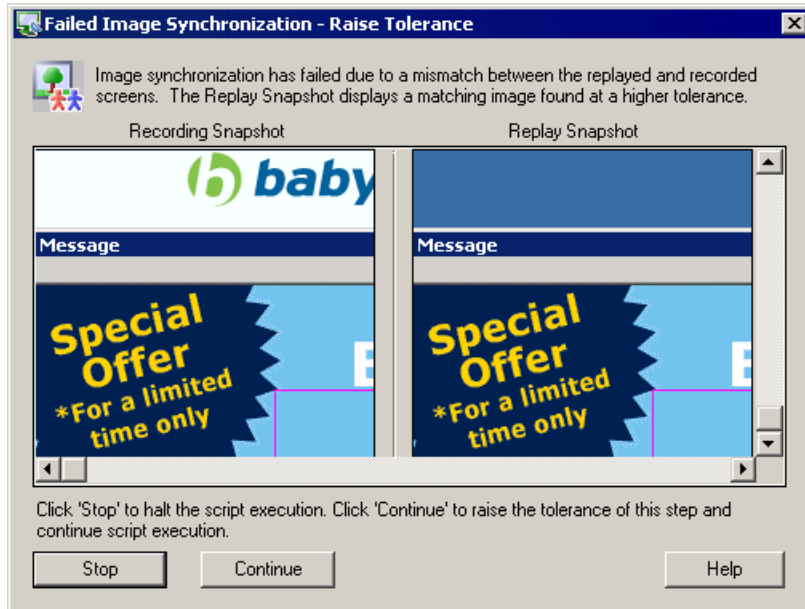
- ▶ **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- ▶ **Continue.** Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.

When you append a new snapshot to the original snapshot, VuGen adds it to the current step. You can then view both the original and appended snapshots by clicking the arrows above the Recording snapshot. The Replay snapshot only shows a single image, the image found during replay.



Raise Tolerance

The Failed Image Synchronization - Raise Tolerance dialog box opens when the script replay failed to find the exact image requested, but if the tolerance level for performing synchronization on images was relaxed, then it would have succeeded in finding the image.

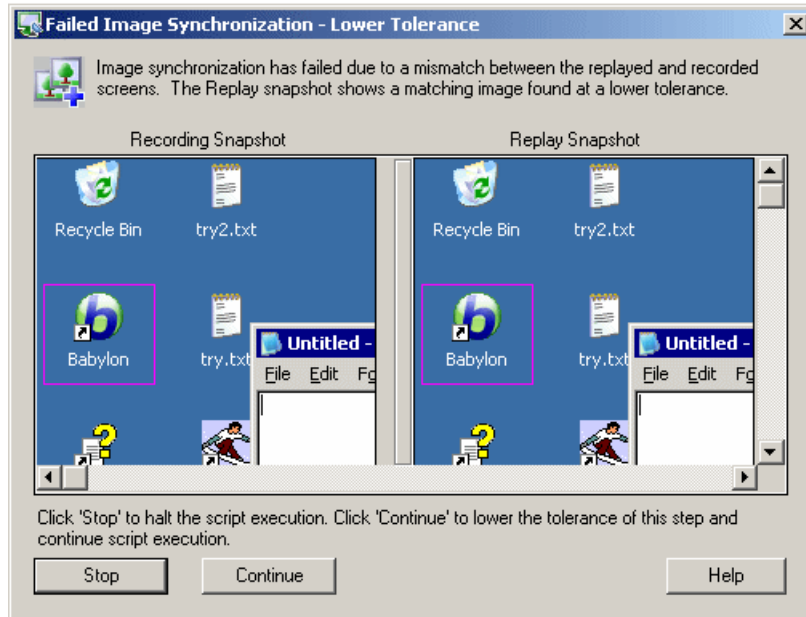


When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch and raise the tolerance level so that VuGen permits a greater degree of a mismatch between the recorded images and those displayed during the replay.

Lower Tolerance

The Failed Image Synchronization - Lower Tolerance dialog box opens when the script replay fails to meet the **NotAppear** or **Change** conditions. VuGen detected an image match where you expected it not to detect one. If the tolerance level was reduced, the recorded and replay images would not match, and the **NotAppear** or **Change** conditions would be met resulting in a successful replay.

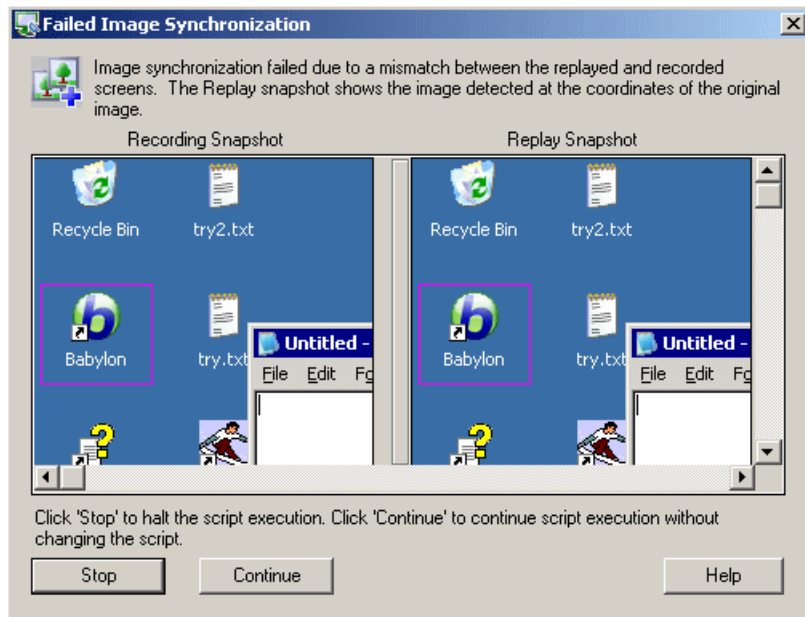


When this dialog box opens, click on one of the following buttons to proceed:

- ▶ **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- ▶ **Continue.** Accept the mismatch and lower the tolerance level so that VuGen permits a less degree of a mismatch between the recorded images and those displayed during the replay.

No Snapshot

The Failed Image Synchronization dialog box opens when the script replay fails to meet any of the synchronization conditions such as **NotAppear** or **Change**. VuGen did not find another image at the original coordinates that could be appended to the script. You can instruct VuGen to continue replay despite the mismatch. This is ideal for situations where you know that the missing image is not essential to the business process.



When this dialog box opens, click on one of the following buttons to proceed:

- **Stop.** Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution.
- **Continue.** Accept the mismatch, and do not make any changes in the script. Continue script execution despite the mismatch.

Shifted Coordinates

When replaying a script, a recorded object may appear at different coordinates on the screen. The object is the same, but its placement has been shifted. For example, during recording a window opened at coordinates (100, 100), but during replay at (200, 250).

In this case, the synchronization point will automatically find the new coordinates without any intervention on your part. It will automatically note the difference of 100 pixels in the horizontal axis and 150 pixels in the vertical axis.

All subsequent mouse operations that are coordinate dependent will use the modified coordinates, so that a mouse click recorded at (130, 130) will be replayed to (230, 280) = (130 + 100, 130 + 150).

You control the shifting of the coordinates through the **AddOffsetToInput** parameter in the **rdp_sync_on_image** step. You can override this parameter to either add or not add the differences in location during replay to the recorded coordinates for any further operations. If you do not override this parameter, VuGen takes its value from the default setting in the run-time settings.

The corresponding parameter in the operations (for example **rdp_mouse_click** or **rdp_mouse_drag**) is **Origin**. This parameter decides whether the operation should take its coordinates only from the 'clean' values that were recorded, or whether it should take into account the differences that were added by the last synchronization point. If not explicitly specified, VuGen takes the value for this parameter from the run-time settings.

21

RDP - Agent for Microsoft Terminal Server

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides you with several important benefits that can enhance your script.

This chapter includes:

- About the Agent for Microsoft Terminal Server on page 351
- Using the Agent for Microsoft Terminal Server Features on page 352
- Installing the Microsoft Terminal Server Agent on page 355
- Effects and Memory Requirements on page 356

About the Agent for Microsoft Terminal Server

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides enhancements to the normal RDP functionality. It is provided in the product's DVD and you can install it on any RDP server.

The agent provides you with the following benefits:

- Intuitive and readable scripts
- Built-in synchronization
- Detailed information about relevant objects

Using the Agent for Microsoft Terminal Server Features

The Agent for Microsoft Terminal Server provides enhancements to the normal RDP functionality. The following sections describe these enhancements and provide details and script samples:

Object Detail Recording

When the Agent for Microsoft Terminal Server is installed, VuGen can record specific information about the object that is being used instead of general information about the action. For example, VuGen generates **Sync Object Mouse Click** and **Sync Object Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows a double-mouse-click action recorded with and without the agent installation. Note that with the agent, VuGen generates `sync_object` functions for all of the mouse actions.

```
rdp_sync_object_mouse_double_click("StepDescription=Mouse Double Click on Synchronized Object 1",
    "Snapshot=snapshot_12.inf",
    "WindowTitle=RDP2",
    "Attribute=TEXT",
    "Value=button1",
    "MouseX=100",
    "MouseY=71",
    "MouseButton=LEFT_BUTTON",
    RDP_LAST);

rdp_mouse_double_click("StepDescription=Mouse Double Click 1",
    "Snapshot=snapshot_2.inf",
    "MouseX=268",
    "MouseY=592",
    "MouseButton=LEFT_BUTTON",
    "Origin=Default",
    RDP_LAST);
```


Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When the agent is not active, you are limited to only inserting **Mouse Click**, **Mouse Double Click**, and **Sync on Image** steps.

When the agent is installed, you are able to insert all possible steps that involve the RDP agent. Below you can find explanations about some of them:

- ▶ **Get Object Info** and **Sync on Object Info**. Provide information about the state of the object and synchronize on a specific object property such as: ENABLED, FOCUSED, CONTROL_ID, ITEM_TEXT, TEXT, CHECKED, and LINES.

In the following example, the `rdp_sync_on_object_info` function provides synchronization by waiting for the Internet Options dialog box to come into focus.

```
rdp_sync_on_object_info("StepDescription=Sync on Object Info 0",
    "Snapshot=snapshot_30.inf",
    "WindowTitle=Internet Options",
    "ObjectX=172",
    "ObjectY=155",
    "Attribute=FOCUSED",
    "Value={valueParam}",
    "Timeout=10",
    "FailStepIfNotFound=No",
    RDP_LAST);
```

Tips for Using the Agent for Microsoft Terminal Server

The following section contains a list of tips and best practises for recording RDP scripts while using the Agent for Microsoft Terminal Server.

- ▶ When opening applications menus (e.g. File, Edit...) with the mouse, sync steps will sometimes fail. To avoid this issue, we recommend selecting menu items by using the keyboard when recording.

- ▶ When you add a **sync_on_object_mouse_click** step manually, the coordinates given are absolute coordinates (relating to the entire screen). To create the sync point, you need to calculate the offset in the window (relative coordinates) of the desired click location and modify the absolute coordinates accordingly for the synchronization to successfully replay.
- ▶ If a synchronization object exists at the correct location and time during replay, but is covered by another window (such as a pop-up), then the synchronization step will pass and a click will be executed on the window which is covering the synchronization point and therefore harm the script flow.
- ▶ During recording, if you want to return the AUT window to the foreground either click on the title bar or use the keyboard (ALT+TAB). If you click inside the AUT window to return it to the foreground, the RDP session may terminate unexpectedly.

Installing the Microsoft Terminal Server Agent

The installation file for the Agent for Microsoft Terminal Server is located on the product installation disk, under the **Additional Components\Agent for Microsoft Terminal Server** directory.

Note that the agent should only be installed on your RDP server machine, not Load Generator machines.

If you are upgrading the agent, make sure to uninstall the previous version before installing the next one (see uninstallation instructions below).

To install the Agent for Microsoft Terminal Server:

- 1 If your server requires administrator permissions to install software, log in as an administrator to the server.
- 2 If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

```
Change user /install
```

- 3 Locate the installation file, **Setup.exe**, on the LoadRunner DVD in the **Additional Components\Agent for Microsoft Terminal Server** directory.
- 4 Follow the installation wizard to completion.

Note: To use the agent, you must set the recording options before recording a Vuser script. In the Start Recording dialog box, click Options. In the Advanced Code Generation node, check **Use RDP Agent**.

To uninstall the Agent for Microsoft Terminal Server:

- 1 If your server requires administrator privileges to remove software, log in as an administrator to the server.

- 2 Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Microsoft Terminal Server** and click **Change/Remove**.

Effects and Memory Requirements

When you run RDP Vusers with the agent installed, each Vuser runs its own process of **lrrdpagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine.

22

Database Protocols

You use VuGen to record communication between a database client application and a server. The resulting script is called a Database Vuser script.

This chapter includes:

- ▶ About Developing Database Vuser Scripts on page 358
- ▶ Introducing Database Vusers on page 359
- ▶ Understanding Database Vuser Technology on page 360
- ▶ Getting Started with Database Vuser Scripts on page 361
- ▶ Using LRD Functions on page 362
- ▶ Understanding Database Vuser Scripts on page 363
- ▶ Working with Grids on page 366
- ▶ Troubleshooting Database Protocols on page 368
- ▶ Evaluating Error Codes on page 369
- ▶ Handling Errors on page 370

About Developing Database Vuser Scripts

When you record a database application communicating with a server, VuGen generates a Database Vuser script. VuGen supports the following database types: CtLib, DbLib, Informix, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity. Each LRD function has an **lrd** prefix and represents one or more database functions. For example, the **lrd_fetch** function represents a fetch operation.

When you run a recorded session, the Vuser script communicates directly with the database server, performing the same operations as the original user. You can set the Vuser behavior (run-time settings) to indicate the number of times to repeat the operation and the interval between the repetitions. For more information, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

Using VuGen, you can parameterize a script, replacing recorded constants with parameters. For more information, see "Creating Parameters" in *Volume I-Using VuGen*.

In addition, you can correlate queries or other database statements in a script, linking the results of one query with another. For more information, see "Correlating Statements" in *Volume I-Using VuGen*.

For troubleshooting information and scripting tips, see the debugging tips.

Introducing Database Vusers

Suppose that you have a database of customer information that is accessed by customer service personnel located throughout the country. You use Database Vusers to emulate the situation in which the database server services many requests for information. A Database Vuser could:

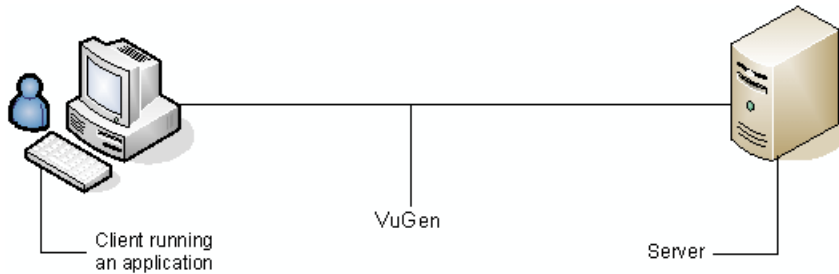
- connect to the server
- submit an SQL query
- retrieve and process the information
- disconnect from the server

You distribute several hundred Database Vusers among the available load generators, each Vuser accessing the database by using the server API. This enables you to measure the performance of your server under the load of many users.

The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. The Vusers execute the script and emulate user load on the client/server system. The Vusers generate performance data which you can analyze in report and graph format.

Understanding Database Vuser Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.



Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.



You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and UNIX environments. For information about recording scripts, see "Recording with VuGen" in *Volume I-Using VuGen*.

Users working in a UNIX only environment can create Database Vuser scripts through programming using VuGen templates as the basis for a script. For information about programming Database Vuser scripts on UNIX, see the appendix "Programming Scripts on UNIX Platforms" in *Volume I-Using VuGen*.

Getting Started with Database Vuser Scripts

This section provides an overview of the process of developing Database Vuser scripts using VuGen.

To develop a Database Vuser script:

1 Record the basic script using VuGen.

Invoke VuGen and create a new Vuser script. Specify the type of Vuser (**Client Server** or **ERP** protocol types). Select an application to record and set the recording options. Record typical operations on your application.

For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same query many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

4 Correlate queries (optional).

Correlating database statements allows you to use the result of a query in a subsequent one. This feature is useful when working on a database with user constraints.

For details, see "Correlating Statements" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser script behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using LRD Functions

The functions developed to emulate communication between a database client and a server are called LRD Vuser functions. Each LRD Vuser function has an **lrd** prefix. VuGen automatically records most of the LRD functions listed in this section during a database session (CtLib, DbLib, Informix, Oracle (2-Tier), and ODBC). You can also manually program any of the functions into your script.

The LRD functions are classified into several categories: Access Management Functions, Environment Functions, Retrieval Handling, Statement Handling, Statement Correlating, Variable Handling, Siebel, and Oracle 8.

For syntax and examples of the LRD functions, see the *Online Function Reference* (**Help > Function Reference**).

Understanding Database Vuser Scripts

After you record a database session, you can view the recorded code in VuGen's built-in editor. You can scroll through the script, see the SQL statements that were generated by your application, and examine the data returned by the server.

The VuGen window provides you with the following information about the recorded database session:

- the sequence of functions recorded
- grids displaying the data returned by database queries
- the number of rows fetched during a query

Function Sequence

When you view a Vuser script in the VuGen window, you see the sequence in which VuGen recorded your activities. For example, the following sequence of functions is recorded during a typical Oracle database session:

lrd_init	Initializes the environment.
lrd_open_connection	Connects to the database server.
lrd_open_cursor	Opens a database cursor.
lrd_stmt	Associates an SQL statement with a cursor.
lrd_bind_col	Binds a host variable to a column.
lrd_exec	Executes an SQL statement.
lrd_fetch	Fetches the next record in the result set.
lr_commit	Commits a database transaction.
lr_close_cursor	Closes a cursor.
lrd_close_connection	Disconnects from the database server.
lrd_end	Cleans up the environment.

In the following script, VuGen recorded the actions of an operator who opened a connection to an Oracle server and then performed a query requesting the local settings.

```
lrd_init(&InitInfo, DBTypeVersion);
lrd_open_connection(&Con1, LRD_DBTYPE_ORACLE, "s1", "tiger", "hp1", "", 0, 0, 0);
lrd_open_cursor(&Csr1, Con1, 0);
lrd_stmt(Csr1, "select parameter, value  from v$nls_parameters "
    " where (upper(parameter) in ('NLS_SORT','NLS_CURRENCY',"
    "'NLS_ISO_CURRENCY', 'NLS_DATE_LANGUAGE',"
    "'NLS_TERRITORY'))", -1, 0 /*Non deferred*/, 1 /*Dflt Ora Ver*/, 0);
lrd_bind_col(Csr1, 1, &D1, 0, 0);
lrd_bind_col(Csr1, 2, &D2, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_fetch(Csr1, 7, 7, 0, PrintRow2, 0);
...
lrd_close_cursor(&Csr1, 0);
lrd_commit(0, Con1, 0);
lrd_close_connection(&Con1, 0, 0);
lrd_end(0);
```

Row Information

VuGen generates an `lrd_fetch` function for each SQL query.

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

The second parameter of the function indicates the number of rows fetched. This number can be positive or negative.

Positive Row Values

A positive value shows the number of rows fetched during recording, and indicates that not all rows were fetched. (For example, if the operator cancelled the query before it was completed.)

In the following example, four rows were retrieved during the database query, but not all of the data was fetched.

```
lrd_fetch(Csr1, 4, 1, 0, PrintRow7, 0);
```

During execution, the script retrieves the number of rows indicated by the positive value (provided the rows exist).

Negative Row Values

A negative row value indicates that all available rows were fetched during recording. The absolute value of the negative number is the number of rows fetched.

In the following example, all four rows of the result set were retrieved:

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

When you execute an **lrd_fetch** statement containing a negative row value, it retrieves all of the available rows in the table at the time of the run—not necessarily the number at the time of recording. In the above example, all four rows of the table were retrieved during the recording session. However, if more rows are available during script execution, they are all retrieved.

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

Working with Grids

The data returned by a query during a recording session is displayed in a grid. By viewing the grid you can determine how your application generates SQL statements and the efficiency of your client/server system.

The data grid is represented by a **GRID** statement. To open the data grid, click on the icon in the margin adjacent to the GRID statement.

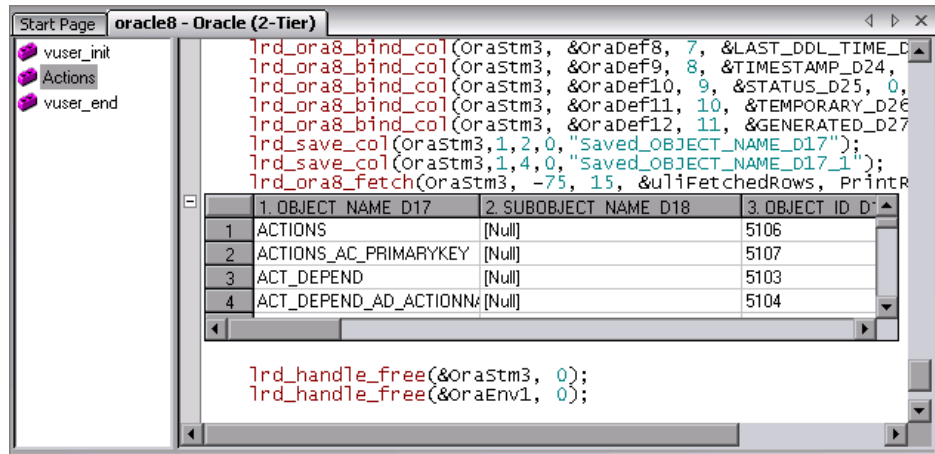
```

Start Page oracle_test - Oracle (2-Tier)
vuser_init
Action
vuser_end

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, &uliRowSP
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, &uliRowSP
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.



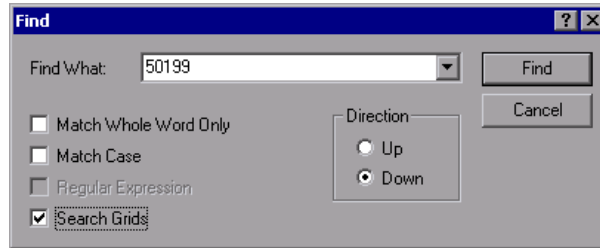
If the data value is very long, only part of it is shown in the grid. This truncation only occurs in the displayed grid and has no impact on the data.

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:

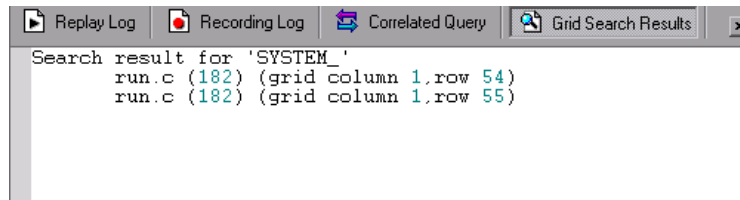
```
[general]
max_line_at_grid=200
```

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.



Troubleshooting Database Protocols

The following section contains troubleshooting information for database protocols.

IE crashes when recording Oracle NCA/11i scripts

This can occur due to an incompatible dll file.

To replace the incompatible dll:

- 1 Open the C:\program files\oracle\JInitiator_1.3.1.18\bin\hotspot directory.
- 2 Back up the **jvm.dll** file.
- 3 Delete the **jvm.dll** file and replace it with a different version of the file.

Evaluating Error Codes

When a Vuser executes an LRD function, the function generates a return code. A return code of 0 indicates that the function succeeded. For example, a return code of 0 indicates that another row is available from the result set. If an error occurs, the return code indicates the type of error. For example, a return code of 2014 indicates that an error occurred in the initialization.

There are four types of return codes, each represented by a numerical range:

Type of Return Code	Range
Informational	0 to 999
Warning	1000 to 1999
Error	2000 to 2999
Internal Error	5000 to 5999

For more detailed information on the return codes, see the *Online Function Reference* (**Help > Function Reference**).

You can evaluate the return code of an LRD function to determine if the function succeeded. The following script segment evaluates the return code of an `lrd_fetch` function:

```
static int rc;
rc=lrd_fetch(Csr15, -13, 0, 0, PrintRow4, 0);
if (rc==0)
    lr_output_message("The function succeeded");
else
    lr_output_message("The function returned an error code:%d",rc);
```

Handling Errors

You can control how database Vusers handle errors when you run a database Vuser script. By default, if an error occurs during script execution, the script execution is terminated. To change the default behavior, you can instruct the Vuser to continue when an error occurs. You can apply this behavior in the following ways:

- ▶ Globally Modifying Error Handling. Provides error handling to the entire script, or to a segment of the script
- ▶ Locally Modifying Error Handling. Provides error handling to a specific function only

Globally Modifying Error Handling

You can change the way that Vusers handle errors by issuing an `LRD_ON_ERROR_CONTINUE` or `LRD_ON_ERROR_EXIT` statement. By default, a Vuser aborts the script execution when it encounters any type of error—database, parameter related, and so on. To change the default behavior, insert the following line into your script:

```
LRD_ON_ERROR_CONTINUE;
```

From this point on, the Vuser continues script execution, even when an error occurs.

You can also specify that the Vuser continue script execution when an error occurs only within a segment of the script. For example, the following code tells the Vuser to continue script execution even if an error occurs in the `lrd_stmt` or `lrd_exec` functions:

```
LRD_ON_ERROR_CONTINUE;
lrd_stmt(Csr1, "select..."...);
lrd_exec(...);
LRD_ON_ERROR_EXIT;
```

Use the `LRD_ON_ERROR_CONTINUE` statement with caution, as significant and severe errors may be missed.

Locally Modifying Error Handling

You can set error handling for a specific function by modifying the severity level. Functions such as `lrd_stmt` and `lrd_exec`, which perform database operations, use severity levels. The severity level is indicated by the function's final parameter, `miDBErrorSeverity`. This parameter tells the Vuser whether or not to continue script execution when a database error occurs (error code 2009). The default, 0, indicates that the Vuser should abort the script when an error occurs.

For example, if you reference a table that does not exist, the following database statement fails and the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
          1 /*Dflt Ora Ver*/, 0);
```

To tell a Vuser to continue script execution, even when a database operation error occurs for that function, change the statement's severity parameter from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1 /*Deferred*/,
          1 /*Dflt Ora Ver*/, 1);
```

When the severity is set to 1 and a database error occurs, a warning is issued. Note that the severity level set for a particular function applies only to that function.

CtLib Result Set Errors

In CtLib recording, the application retrieves all of the available result sets after executing a statement. If the returned result set contains fetchable data, the application performs bind and fetch operations on the data as indicated in the following example:

```
lrd_stmt(Csr15, "select * from all_types", -1, 148, -99999, 0);
lrd_exec(Csr15, 0, 0, 0, 0, 0);
lrd_result_set(Csr15, 1 /*Succeed*/, 4040 /*Row*/, 0);
lrd_bind_col(Csr15, 1, &tinyint_D41, 0, 0);
...
lrd_fetch(Csr15, -9, 0, 0, PrintRow3, 0);
```

If a result set does not contain fetchable data, bind and fetch operations cannot be performed.

When you parameterize your script, result data may become unfetchable (depending on the parameters). Therefore, a CtLib session that recorded bind and fetch operations for a particular statement, may not be able to run, if the new data is unfetchable. If you try to execute an **lrd_bind_col** or an **lrd_fetch** operation, an error will occur (LRDRET_E_NO_FETCHABLE_DATA — error code 2064) and the Vuser will terminate the script execution.

You can override the error by telling the Vuser to continue script execution when this type of error occurs. Insert the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_CONT;
```

To return to the default mode of terminating the script execution, type the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_EXIT;
```

Use this option with caution, as significant and severe errors may be missed.

23

Database - Script Correlation

After you record a database session, you may need to correlate one or more queries within your script—use a value that was retrieved during the database session, at a later point in the session.

This chapter includes:

- About Correlating Database Vuser Scripts on page 373
- Scanning a Script for Correlations on page 374
- Correlating a Known Value on page 376
- Database Correlation Functions on page 378

About Correlating Database Vuser Scripts

If you encounter an error when running your script, examine the script at the point where the error occurred. In many cases, you can overcome the problem by correlating the query. Correlating the query means that you save a run-time value to a parameter. You then use the saved value at a later point in the same script. In summary, correlation is using the results of one statement as input to another.

There are many queries whose inputs depend on the result of prior queries. To emulate this behavior, use VuGen's correlation capabilities.

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script and allow a successful replay. It performs the following steps:

- ▶ Scans for potential correlations
- ▶ Inserts the appropriate correlation function to save the results to a parameter
- ▶ Replaces the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

To scan and correlate a script detected with automatic correlation:

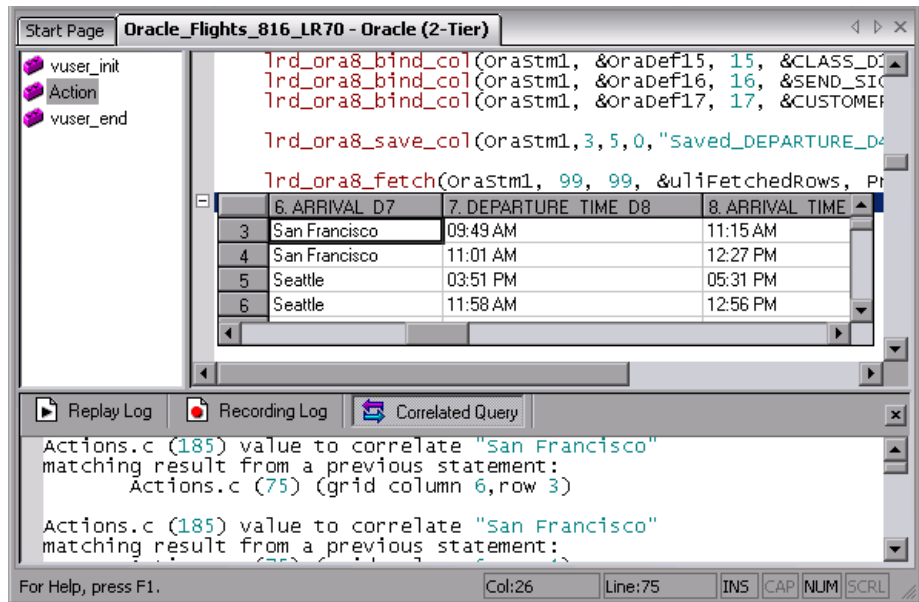
- 1** Open the Output window.

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

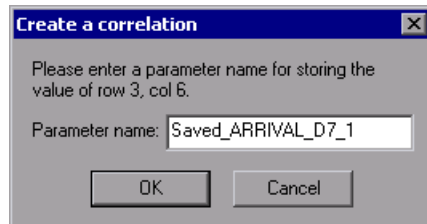
- 2** Select **Vuser > Scan for Correlations**.

VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.

In the following example, in the `lrd_ora8_fetch` function, VuGen detected a value to correlate.



- 3 In the Correlated Query tab, double-click on the result you want to correlate. Click on the words **(grid column x, row y)** VuGen sends the cursor to the location of the value in your grid.
- 4 Select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



- 5 Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (`lrd_save_value`, `lrd_save_col`, or `lrd_save_ret_param`, `lrd_ora8_save_col`) which saves the result to a parameter.

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

A message box opens asking if you want to search for all occurrences of the value in the script.

- To replace only the value in the selected statement, click **No**.
 - To search and replace additional occurrences, click **Yes**.
- 7 A Search and Replace dialog box opens. Confirm any replacements, including your original statement.
 - 8 Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. If you cancel the correlation, VuGen also erases the statement created in the previous step.

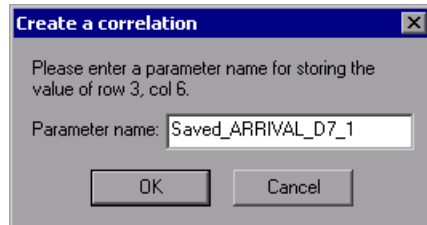
Correlating a Known Value

If you know which value needs to be correlated, perform the following procedure.

To correlate a specific value:

- 1 Locate the statement in your script, with the query containing the value you want to correlate. This is usually one of the arguments of the **lrd_assign**, **lrd_assign_bind**, or **lrd_stmt** functions. Select the value without the quotation marks.
- 2 Select **Scan for Correlations (at cursor)** from the right-click menu. VuGen scans the selected value for correlations.
- 3 In the Output window's **Correlated Query** tab, double-click on the result you want to correlate. Click on the words (**grid column x, row y**). VuGen sends the cursor to the location of the value in your grid.

- 4 In the grid, click on the value you want to correlate and select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



- 5 Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**, **lrd_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).

Database Correlation Functions

When working with Database Vuser scripts, (DbLib, CtLib, Oracle, Informix, and so forth) you can use VuGen's automated correlation feature to insert the appropriate functions into your script. The correlating functions are:

- ▶ **lrd_save_col** saves a query result appearing in a grid, to a parameter. This function is placed before fetching the data. It assigns the value retrieved by the subsequent **lrd_fetch** to the specified parameter. (**lrd_ora8_save_col** for Oracle 8 and higher)
- ▶ **lrd_save_value** saves the current value of a placeholder descriptor to a parameter. It is used with database functions that set output placeholders (such as certain stored procedures under Oracle).
- ▶ **lrd_save_ret_param** saves a stored procedure's return value to a parameter. It is used primarily with database procedures stored in DbLib that generate return values.

Note: VuGen does not apply correlation if the saved value is invalid or NULL (no rows returned).

For more information about these functions and their arguments, see the *Online Function Reference* (**Help > Function Reference**).

24

DNS Protocol

VuGen allows you to emulate network activity by directly accessing a DNS server.

This chapter includes:

- ▶ About Developing DNS Vuser Scripts on page 379
- ▶ Working with DNS Functions on page 380

About Developing DNS Vuser Scripts

The DNS protocol is a low-level protocol that allows you to emulate the actions of a user working against a DNS server.

The DNS protocol emulates a user accessing a Domain Name Server to resolve a host name with its IP address. Only replay is supported for this protocol—you need to manually add the functions to your script.

To create a script for the DNS protocol, select **File > New** to open the New Virtual User dialog box. Select the Domain Name Resolution (DNS) protocol type from the Client/Server category. Since recording is not supported for DNS, you program the script with the appropriate DNS, Vuser API and C functions. For more information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

After you create a Vuser script, you integrate it into a scenario on either a Windows or UNIX platform. For more information on integrating Vuser scripts in a scenario, see the *HP LoadRunner Controller User's Guide*.

Working with DNS Functions

DNS Vuser script functions record queries to and from a Domain Name Resolution (DNS) server. Each DNS function begins with a **dns** prefix. For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

Function Name	Description
ms_dns_query	Resolves the IP address of a host.
ms_dns_nextresult	Advances to the next IP address in the list returned by ms_dns_query .

In the following example, a query is submitted to the DNS server and the results are printed to the log file.

```

Actions()
{
int  rescnt = 0;
char results = NULL;
results = (char *) ms_dns_query("transaction",
                                "URL=dns://<DnsServer>",
                                "QueryHost=<Hostname>",
                                LAST);

// List all the IP addresses of the host names...
while (*results) {
    rescnt++;
    lr_log_message(lr_eval_string("(%d) IP of<Hostname> is %s"),
                  rescnt, results);
    results = (char *) ms_dns_nextresult(results);
}
return 1;
}

```

25

Windows Sockets (WinSock) Protocol

You use VuGen to record communication between a client application and a server that communicate using the Windows Sockets protocol. The resulting script is called a Windows Sockets Vuser script.

This chapter includes:

- About Recording Windows Sockets Vuser Scripts on page 382
- Getting Started with Windows Sockets Vuser Scripts on page 383
- Setting the WinSock Recording Options on page 384
- Using LRS Functions on page 387
- Working with Windows Socket Data on page 388
- Viewing Data in the Snapshot Window on page 389
- Navigating Through the Data on page 391
- Modifying Buffer Data on page 394
- Modifying Buffer Names on page 401
- Viewing Windows Socket Data in Script View on page 402
- Understanding the Data File Format on page 403
- Viewing Buffer Data in Hexadecimal format on page 405
- Setting the Display Format on page 408
- Debugging Tips on page 411
- Manually Correlating WinSock Scripts on page 412

About Recording Windows Sockets Vuser Scripts

The Windows Sockets protocol is ideal for analyzing the low level code of an application. For example, to check your network, you can use a Windows Sockets (WinSock) script to see the actual data sent and received by the buffers. The WinSock type can also be used for recording other low level communication sessions. In addition, you can record and replay applications that are not supported by any of the other Vuser types.

When you record an application which uses the Windows Sockets protocol, VuGen generates functions that describe the recorded actions. Each function begins with an **Lrs** prefix. The LRS functions relate to the sockets, data buffers, and the Windows Sockets environment. Using VuGen, you record your application's API calls to the `Winsock.dll` or `Wsock32.dll`.

For example, you could create a script by recording the actions of a *telnet* application.

In the following example, `Lrs_send` sends data to a specified socket:

```
Lrs_send("socket22", "buf44", LrsLastArg);
```

You can view and edit the recorded script from VuGen's main window. The Windows Sockets API calls that were recorded during the session are displayed in the window, allowing you to track your network activities.

VuGen can display a WinSock script in two ways:

- ▶ As an icon-based representation of the script. This is the default view, and is known as the **Tree view**.
- ▶ As a text-based representation of the script showing the Windows Sockets API calls. This is known as the **Script view**.

You use VuGen to view and edit your Vuser script from either the Tree view or Script view. For more information, see "Introducing Service TestVuGen" in *Volume I-Using VuGen*.

After creating a script, you can view the recorded data as a snapshot or as a raw data file. For details, see "Working with Windows Socket Data" on page 388

Getting Started with Windows Sockets Vuser Scripts

This section provides an overview of the process of developing Windows Sockets Vuser scripts using VuGen.

To develop a Windows Sockets script:

1 Record the actions using VuGen.

Invoke VuGen and create a new Vuser script, specifying Windows Sockets as the type. Select an application to record and set the recording options. Record typical operations on your application.

For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see "Correlating Statements" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" and "Configuring Network Run-Time Settings" in *Volume I-Using VuGen*.

6 Run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

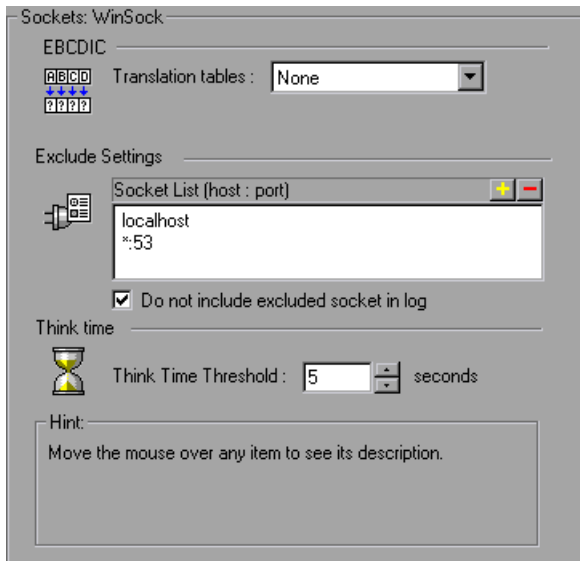
After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Setting the WinSock Recording Options

The following recording options are available for WinSock Vusers:

- ▶ Configuring the Translation Table
- ▶ Excluding Sockets
- ▶ Setting the Think Time Threshold

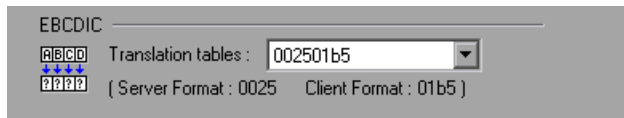
To open the Recording Options dialog box, select **Tools > Recording Options** or click the **Options** button in the Start Recording dialog box. VuGen displays the WinSock options.



Configuring the Translation Table

To display data in EBCDIC format, you specify a translation table in the recording options.

The Translation Table lets you specify the format for recording sessions. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. Select a translation option from the list box.



The first four digits of the listbox item represent the server format. The last four digits represent the client format. In the above example, the selected translation table is 002501b5. The server format is 0025 and the client format is 01b5 indicating a transfer from the server to the client. In a transmission from the client to the server, you would select the item that reverses the formats—01b50025 indicating that the client's 01b5 format needs to be translated to the server's 0025 format.

The translation tables are located in the **ebcdic** directory under the VuGen's installation directory. If your system uses different translation tables, copy them to the **ebcdic** directory.

Note: If your data is in ASCII format, it does not require translation. You must select the **None** option, the default value. If you do select a translation table, VuGen will translate the ASCII data.

When working on Solaris machines, you must set the following environment variables on all machines running the Vuser scripts:

```
setenv LRSDRV_SERVER_FORMAT 0025
setenv LRSDRV_CLIENT_FORMAT 04e4
```

Excluding Sockets

VuGen supports the Exclude Socket feature, allowing you to exclude a specific socket from your recording session. To exclude all actions on a socket from your script, you specify the socket address in the Exclude Socket list. To add a socket to the list, click the plus sign in the upper right corner of the box and enter the socket address in one of the following formats:

Value	Meaning
host:port	Exclude only the specified port on the specified host.
host	Exclude all ports for the specified host.
:port	Exclude the specified port number on the local host.
*:port	Exclude the specified port number on all hosts.

You can exclude multiple hosts and ports by adding them to the list. To remove a socket from the excluded list, select the socket address and click the minus sign in the upper right corner of the box. We recommend that you exclude hosts and ports that do not influence the server load under test, such as the local host and the DNS port (53), which are excluded by default.

By default, VuGen does not log the actions of the excluded sockets in the Excluded Socket List. To instruct VuGen to log the actions of the excluded socket(s) clear the Do not include excluded sockets in log check box. When logging is enabled for the excluded sockets, their actions are preceded by "Exclude" in the log file.

```
Exclude : /* recv(): 15 bytes were received from socket 116 using flags 0 */
```

Setting the Think Time Threshold

During recording, VuGen automatically inserts the operator's think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an **lr_think_time** statement before LRS functions. If the recorded think time is below the threshold level, an **lr_think_time** statement is not generated.

To set the think time threshold, enter the desired value (in seconds) in the **Think Time Threshold** box. The default value is five seconds.

Using LRS Functions

The functions developed to emulate communication between a client and a server by using the Windows Sockets protocol are called LRS Vuser functions. Each LRS Vuser function has an **lrs** prefix. VuGen automatically records most of the LRS functions listed in this section during a Windows Sockets session. You can also manually program any of the functions into your Vuser script.

The LRS functions are classified into several categories: Socket, Buffer, Environment, Correlating Statements, Conversion, and Timeout.

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

After you record a session, VuGen's built-in editor lets you view the recorded code. You can scroll through the script, view the functions that were generated by your application, and examine the transferred data. When you view the script in the main window, you see the sequence in which VuGen recorded your activities.

The following function sequence is recorded during a typical session:

lrs_startup	Initializes the WinSock DLL.
lrs_create_socket	Initializes a socket.
lrs_send	Sends data on a datagram or to a stream socket.
lrs_receive	Receives data from a datagram or stream socket.
lrs_disable_socket	Disables an operation on a socket.
lrs_close_socket	Closes an open socket.
lrs_cleanup	Terminates the use of the WinSock DLL.

VuGen supports record and replay for applications using the Windows Socket protocol on Windows; on UNIX platforms, only replay is supported.

Working with Windows Socket Data

After you record an application using VuGen, you have multiple data buffers containing the data.

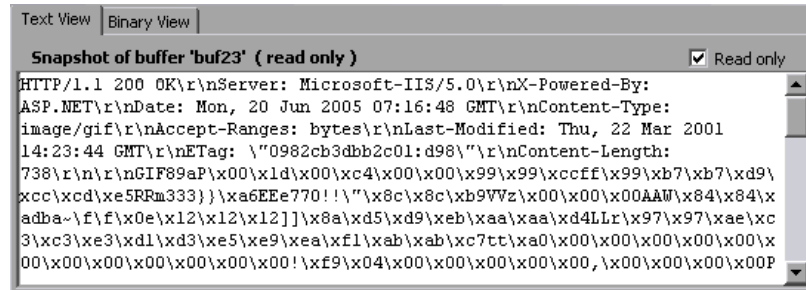
When you view the WinSocket script in tree view, VuGen provides a snapshot window which allows you to navigate within the data buffers and modify the data.

When working in script view, you can view the raw data in the **data.ws** file. For more information, see "Viewing Windows Socket Data in Script View" on page 402.

Viewing Data in the Snapshot Window

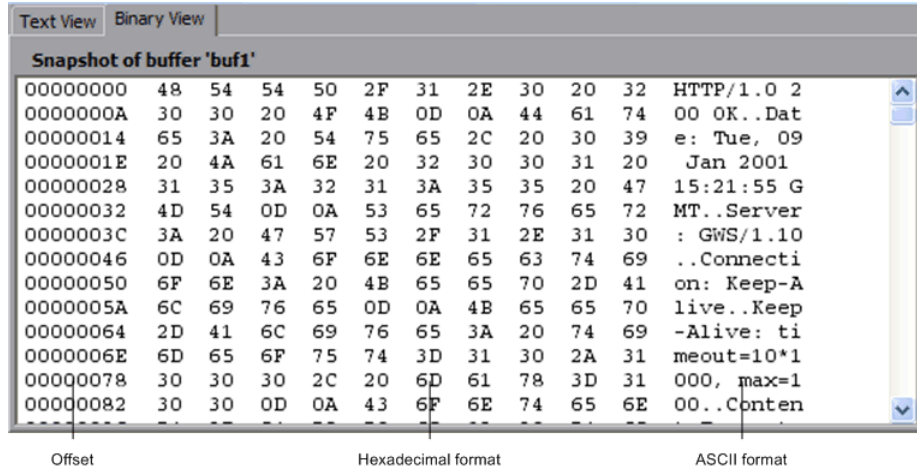
When viewing a Windows Socket script in tree view, VuGen provides a buffer snapshot window which displays the data in an editable window. You can view a snapshot in either **Text** view or **Binary** view.

The text view shows a snapshot of the buffer with the contents represented as text.



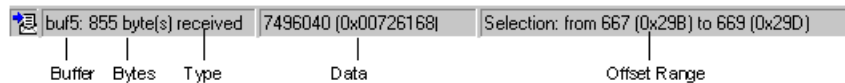
By default, VuGen stores the buffer data as read-only data. If you want to modify the contents of the buffer, clear the **Read only** box in the buffer's Text View. VuGen issues a warning that bookmarks and parameters may be affected.

The binary view shows the data in hexadecimal representation. The left column shows the offset of the first character in that row. The middle column displays the hexadecimal values of the data. The right column shows the data in ASCII format.



The status bar below the buffer snapshot provides information about the data and buffer:

- ▶ **Buffer number.** The buffer number of the selected buffer.
- ▶ **Total bytes.** the total number of bytes in the buffer.
- ▶ **Buffer type.** the type of buffer—received or sent.
- ▶ **Data.** the value of the data at the cursor in decimal and hexadecimal formats, in Little Endian order (reverse of how it appears in the buffer).
- ▶ **Offset.** the offset of the selection (or cursor in text view) from the beginning of the buffer. If you select multiple bytes, it indicates the range of the selection.



The status bar also indicates whether or not the original data was modified.



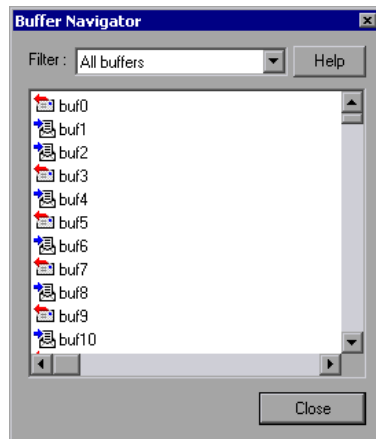
Navigating Through the Data

In tree view, VuGen provides several tools that allow you to navigate through the data in order to identify and analyze a specific value:

- Buffer Navigator
- Go To Offset
- Bookmarks

Buffer Navigator

By default, VuGen displays all the steps and buffers in the left pane. The Buffer Navigator is a floating window that lets you display only the receive and send buffers steps (**lrs_send**, **lrs_receive**, **lrs_receive_ex**, and **lrs_length_receive**). In addition, you can apply a filter and view either the send or receive buffers.



When you select a buffer in the navigator, its contents are displayed in the buffer snapshot window.

If you change a buffer's name after recording, its contents will not appear in the snapshot window when you click on the step. To view the renamed buffer's data, use the buffer navigator and select the new buffer's name. VuGen issues a warning message indicating that parameter creation will be disabled for the selected buffer.

To open the Buffer Navigator, select **View > Buffer Navigator**. To close the navigator, click the X in the top right corner of the navigator dialog box.

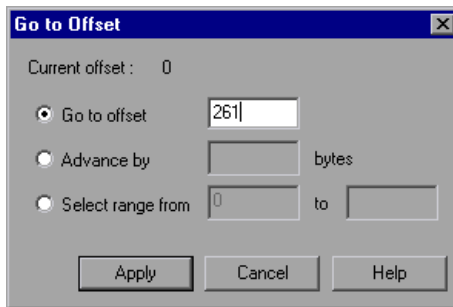
Note that you can also navigate between buffers by clicking on the buffer step in the left pane's tree view. The advantages of the buffer navigator are that it is a floating window with filtering capabilities.

Go To Offset

You can move around within the data buffer by specifying an offset. You can indicate the absolute location of the data, or a location relative to the current position of the cursor within the buffer. This dialog box also lets you select a range of data, by specifying the starting and end offsets.

To go to an offset:

- 1 Click within the snapshot window. Then select **Go to offset** from the right-click menu. The Go to offset dialog box opens.

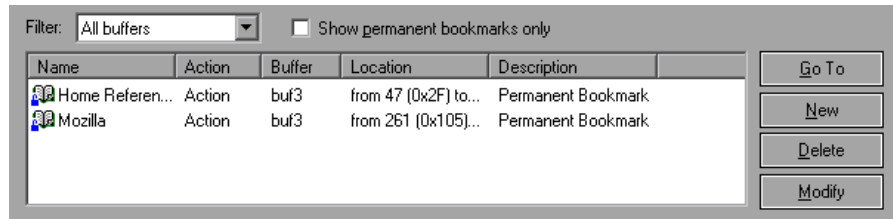


- 2 To go to a specific offset within the buffer (absolute), click **Go to offset** and specify an offset value.
- 3 To jump to a location relative to the cursor, click **Advance by** and specify the number of bytes you want to advance. To advance ahead, enter a positive value. To move backwards within the buffer, use a negative value.

- 4 To select a range of data within the buffer, click **Select range from** and specify the beginning and end offsets.

Bookmarks

VuGen lets you mark locations within a buffer as bookmarks. You give each bookmark a descriptive name and click on it to jump directly to its location. The bookmarks are listed in the Output window's **Bookmarks** tab below the buffer snapshot.



Bookmarks can be used in both the text and binary views. You can locate the desired data in text view, save the location as a bookmark, and jump directly to that bookmark in binary view.

The bookmark can mark a single byte or multiple bytes. When you click on a bookmark in the list, it is indicated in the buffer snapshot window as a selection. Initially, in the text view the data is highlighted in blue, and in binary view the bookmark block is marked in red. Also in binary view, when you place your cursor over a bookmark, a popup text box opens indicating the name of the bookmark.

You can create both permanent and simple bookmarks. A permanent bookmark is always marked within the buffer's binary view—it is enclosed by a blue box. The bookmark stays selected in blue, even when pointing to another location in the buffer. The cursor location is marked in red. A simple bookmark, however, is not permanently marked. When you jump to a simple bookmark, it is marked in red, but once you move the cursor within the buffer, the bookmark is no longer selected. By default bookmarks are permanent.

To work with bookmarks:

- 1** To create a bookmark, select one or more bytes in a buffer snapshot (text or binary view) and select **New Bookmark** from the right-click menu.
- 2** To view the bookmark list, select **View > Output Window** and select the **Bookmarks** tab.
- 3** To assign a name to a bookmark, click on it in the bookmark list and edit the title.
- 4** To change the location of a bookmark, select the bookmark in the **Bookmarks** tab, then select the new data in the buffer snapshot. Click **Modify** in the **Bookmarks** tab.
- 5** To change a bookmark 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, perform a right-click, and clear the check adjacent to **Permanent Bookmark**.
- 6** To display only permanent bookmarks in the list, select the **Show Permanent Bookmarks only** check box in the **Bookmarks** tab.
- 7** To view bookmarks from a specific buffer, select a bookmark from the desired buffer and select **Selected buffer only** in the **Filter** box.
- 8** To delete a bookmark, select it in the **Bookmarks** tab and click **Delete**.

Modifying Buffer Data

In tree view, VuGen provides several tools that allow you to modify the data by deleting, changing or adding to the existing data.

- Inserting Data
- Editing Data
- Parameterizing the Data

Inserting Data

You can insert a numerical value into a data buffer. You can insert it as a single, double-byte or 4-byte value.

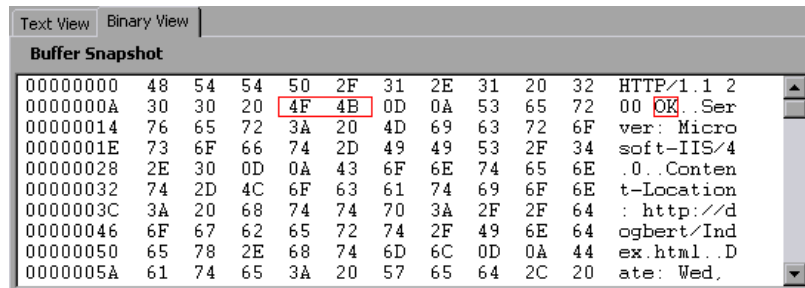
To insert a number into a data buffer:

- 1 Click at a location in the buffer.
- 2 Open the right-click menu and select **Advanced > Insert Number > Specify...**
- 3 Enter the ASCII value that you want to insert into the **Value** box.
- 4 Select the size of the data you want to insert: 1 byte, 2 bytes, or 4 bytes from the **Size** box.
- 5 Click **OK** to finish. VuGen inserts the hexadecimal representation of the data into the buffer.

Editing Data

You can perform all of the standard edit operation on buffer data: copy, paste, cut, delete, and undo. In the binary view you can specify the actual data to insert. VuGen allows you to specify the format of the data—single byte, 2-byte, or 4-byte, and hexadecimal or decimal value. You can copy binary data and insert it as a number into the buffer. You can see the decimal or hexadecimal numbers in the right column of the binary view.

In the following example, the word **OK** was selected.



If you perform simple copy (CTRL+C) and paste (CTRL+V) operations at the beginning of the next line of data, it inserts the actual text.

```
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.

To edit buffer data in the binary view:

1 To copy buffer data:

- As characters, select one or more bytes and press CTRL+C.
- As a decimal number, **Advanced > Copy As Number > Decimal** in the right-click menu.
- As a hexadecimal number, **Advanced > Copy As Number > Hexadecimal** in the right-click menu.

2 To paste the data:

- As a single byte (assuming the size of the data on the clipboard is a single byte), click at the desired location in the buffer and press CTRL+V.
- In short format (2-byte), **Advanced > Insert Number > Paste Short (2-byte)** in the right-click menu.
- In long format (4-byte), **Advanced > Insert Number > Paste Long (4-byte)** in the right-click menu.

- 3 To delete data, select it in either one of the views and select **Delete** from the right-click menu.

Parameterizing the Data

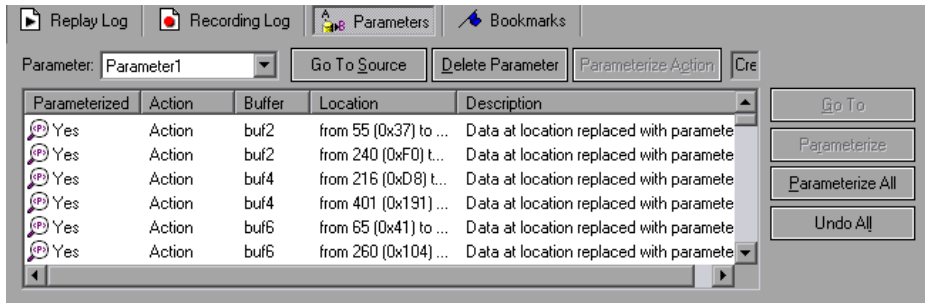
In tree view, VuGen lets you parameterize the data directly from the buffer snapshot view. You can specify a range of what to parameterize and you can specify borders. If you do not specify borders for the parameterized string, then VuGen inserts an **lrs_save_param** function into your script. If you specify borders, VuGen inserts **lrs_save_searched_string** into your script since this function allows you to specify boundary arguments.

Note that the **lrs_save_param** and **lrs_save_searched_string** functions correlate the data. This means that it stores the data that is received, for use in a later point within the test. Since correlation stores the received data, it only applies to Receive buffers and not to Send buffers. The recommended procedure is to select a string of dynamic data within the Receive buffer that you want to parameterize. Use that same parameter in a subsequent Send buffer.

This type of correlation should not be confused with simple parameterization. Simple parameterization (**Insert > New Parameter**) only applies to data within Send buffers. You set up a parameter and assign it several values. VuGen uses the different values in each of the test runs or iterations. For more information, see "Creating Parameters" in *Volume I-Using VuGen*.

The next sections discuss the correlation of data in Receive buffers.

After you create a parameter, VuGen lists all the locations in which it replaced the string with a parameter. VuGen also provides information about the creation of the parameter—the buffer in which it was created and the offset within the buffer. It lists all occurrences of the parameter in the Output Window's **Parameters** tab, below the snapshot view.

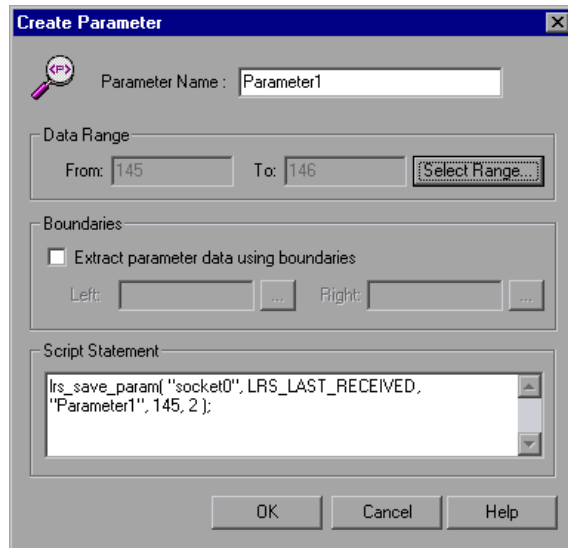


VuGen allows you to manipulate the parameters:

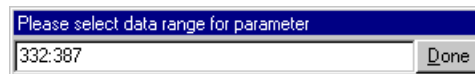
- ▶ **Filtering.** You can filter the parameter replacements by the parameter name.
- ▶ **Go to Source.** Select a replacement and click **Go To Source** to jump to the exact location of the replaced parameter within the buffer.
- ▶ **Deleting.** You can delete any one of the parameters. When you delete a parameter, VuGen replaces the data with its original value and removes the parameterization function from the script.
- ▶ **Name.** You can provide a name to each replacement.
- ▶ **Undo Replacement.** You can also undo one or more replacements displayed in the list.

To parameterize data from the snapshot window:

- 1 Select the data you want to parameterize and select **Create Parameter** from the right-click menu (only available for Receive buffers). A dialog box opens:



- 2 Specify a name for the parameter in the **Parameter Name** box.
- 3 Select a range of characters to parameterize. By default, VuGen takes the range of data that you selected in the buffer. To select a range other than the one that appears in the dialog box, click **Select Range**. A small dialog box opens indicating the selected range.



Select a range in the buffer snapshot window and click **Done**.

- 4 If the parameter data is not constant but its borders are consistent, you can specify a right and left boundary.

To specify boundaries:

- ▶ Select the **Extract Parameter Data Using Boundaries** check box. VuGen changes the function in the **Script Statement** section from `Irs_save_param` to `Irs_save_searched_string`. Click **Done**.
 - ▶ Click the browse button adjacent to the **Left** box in the **Boundaries** section. A small dialog box opens, indicating your selection within the buffer. Select the boundaries within the buffer and click **Done**. Repeat this step for the right boundary.
- 5** Make the desired modifications to the arguments in the **Script Statement** section. For example you can add `_ex` to the `Irs_save_param` function to specify an encoding type. For more information about these functions see the *Online Function Reference* (**Help > Function Reference**).
 - 6** Click **OK** to create the parameter. VuGen asks you for a confirmation before replacing the parameter. Click **Yes**. You can view all the replacements in the **Parameters** tab.
 - 7** To jump to the original location of the parameter within its buffer, select it and click **Go To Source**.
 - 8** To jump to the buffer location of the selected replacement, select it and click **Go To**.
 - 9** To delete an entire parameter, select the parameter in the **Filter** box and click **Delete Parameter**.
 - 10** To undo a replacement, select it in the **Parameters** tab and click **Undo**. To undo all replacements of the displayed parameter, select it in the **Parameters** tab and click **Undo All**.
 - 11** When you undo specific replacements, the Parameterized column shows **No** for that occurrence. To reapply the parameterization rule to an occurrence that was undone, select it and select **Replace With Parameter** from the right-click menu.
 - 12** To delete an entire parameter and undo all the replacements, select the parameter in the **Filter** box and click **Delete Parameters**.
 - 13** Select **Vuser > Parameter List** to assign data to the parameters.

Modifying Buffer Names

You can modify the name of a buffer using the Script view of the **data.ws** file. If you modify a buffer name after recording, this will affect the replay of the Vuser script. You can view the contents of the renamed buffer in the Script view or in Tree view using the Buffer Navigator.

If you created bookmarks in the buffer and it is not longer available, VuGen prompts you to delete the bookmarks within the buffer in which they were defined.

If you created parameters in the buffer and it is not longer available, VuGen prompts you to delete the parameters from the buffer in which they were defined. When you delete the parameter, all replacements are undone, even those in other buffers.

When you view the renamed buffer in the Buffer Navigator, VuGen warns you that parameter creation will be disabled within that buffer.

Viewing Windows Socket Data in Script View

When you use VuGen to create a Windows Sockets Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**. In addition to the Vuser script, VuGen also creates a data file, **data.ws** that contains the data that was transmitted or received during the recording session. You can use VuGen to view the contents of the data file by selecting **data.ws** in the Data Files box of the main VuGen window.

The option to view a data file is available by default for Windows Sockets scripts. Note that you can only view the data in script view.

```

;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: ASPSESSIONID4ACC0B7=DDG9PCHTBAMEL1MNOHALDHATT:

```

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 `Irs_receive` function was recorded during a Vuser session:

```
Irs_receive("socket1", "buf4", LrsLastArg)
```

In this example, `Irs_receive` handled data that was received on `socket1`. The data was stored in the fifth receive record (`buf4`)—note that the index number is zero-based. The corresponding section of the `data.ws` file shows the buffer and its contents.

```
recv buf4 39
"\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
"\r\n"
"\r\n"
"SunOS UNIX (sunny)\r\n"
"\r"
"\x0"
"\r\n"
"\r"
"\x0"
```

Understanding the Data File Format

The `data.ws` data file has the following format:

- File header
- A list of buffers and their contents

The file header includes an internal version number of the data file format. The current version is 2. If you try to access data from a data file with format version 1, VuGen issues an error.

```
;WSRData 2 1
```

An identifier precedes each record, indicating whether the data was received or sent, followed by the buffer descriptor, and the number of bytes received (for `Irs_receive` only). The buffer descriptor contains a number identifying the buffer.

For example,

```
recv buf5 25
```

indicates that the buffer contains data that was received. The record number is 5, indicating that this receive operation was the sixth data transfer (the index is zero based), and twenty-five bytes of data were received.

If your data is in ASCII format, the descriptor is followed by the actual ASCII data that was transferred by the sockets.

If your data is in EBCDIC format, it must be translated through a look-up table. For information on setting the translation table, see "Setting the WinSock Recording Options" on page 384. 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\xff\xfb\x01\xff\xff\x03\xff\xfd\x01"
"\r\n"
"SunOS UNIX (sunny)\r\n"
```

The following segment shows the header, descriptors, and data in a typical data file:

```
;WSRData 2 1

send buf0
"\xff\xfd\x01\xff\xfd\x03\xff\xfb\x03\xff\xfb\x18"

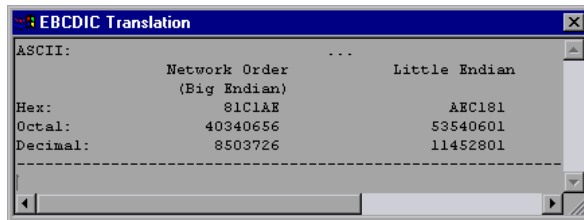
recv buf1 15
"\xff\xfd\x18\xff\xfd\x1f\xff\xfd"
"##"
"\xff\xfd"
""
"\xff\xfd"
"$"

send buf2
"\xff\xfb\x18"
```

Viewing Buffer Data in Hexadecimal format

VuGen contains a utility allowing you to view a segment of data, displaying it in hexadecimal and ASCII format, while indicating the offset of the data.

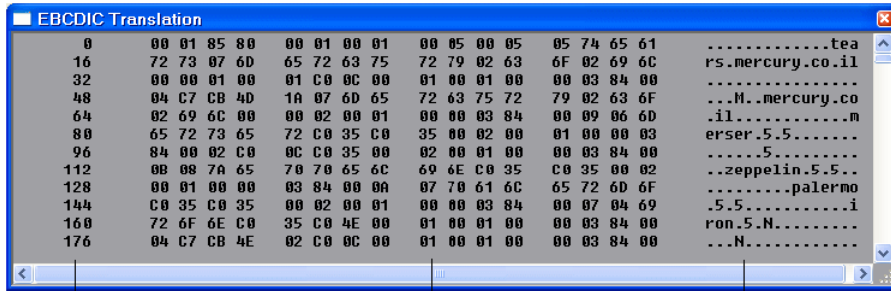
To display the data in the viewer window, select the data and press F7. If the selected text is less than four characters, VuGen displays the data in **short format**, showing the hexadecimal, decimal and octal representations.



You can customize the short format in the **conv_frm.dat** file as described in "Setting the Display Format" on page 408.

If the selected text is more than four characters, VuGen displays the data in several columns in **long format**. You can customize the long format by modifying the **conv_frm.dat** file, as described in "Setting the Display Format" on page 408.

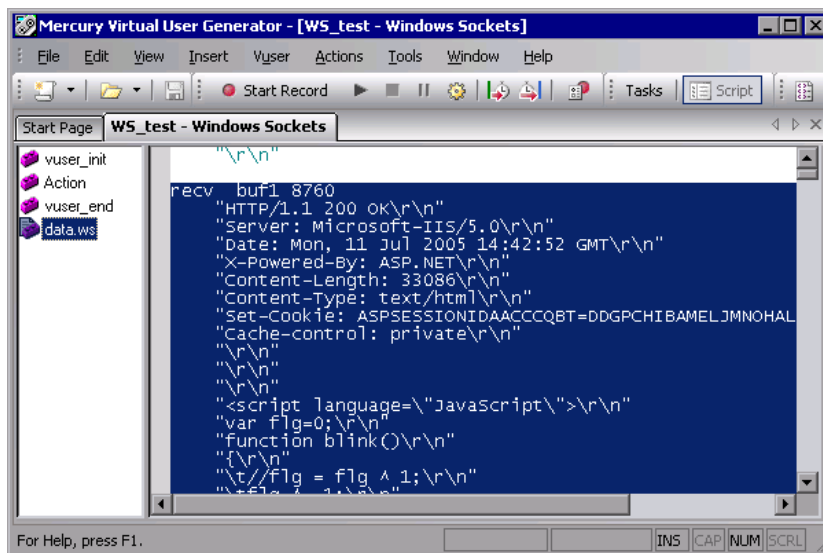
In the default format, the first column displays the character offsets from the beginning of the marked section. The second column displays the hexadecimal representation of the data. The third column shows the data in ASCII format. When displaying EBCDIC data, all non-printable ASCII characters (such as /n), are represented by dots.



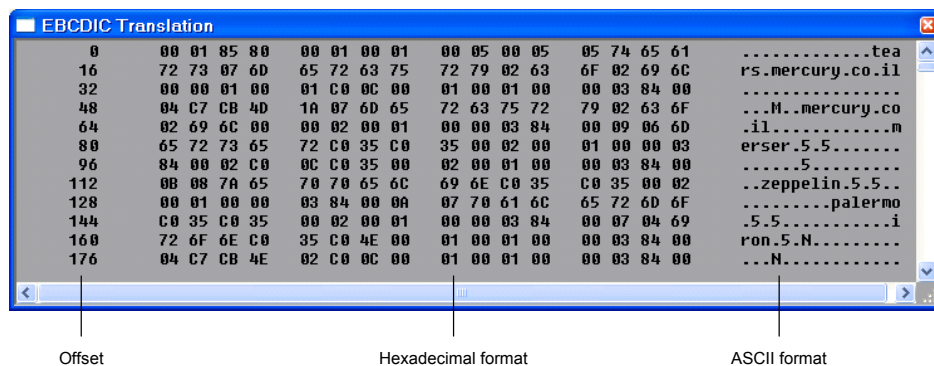
The F7 viewer utility is especially useful for parameterization. It allows you to determine the offset of the data that you want to save to a parameter.

To determine the offset of a specific character:

- 1 View **data.ws** and select the data from the beginning of the buffer.



- 2 Press F7 to display the data and the character offsets. Since more than four characters were selected, the data is displayed in long format.



- 3 Locate the value you want to correlate in the ASCII data. In this example, we will correlate the number 13546 (a process ID during a UNIX session) which begins at the 31st character—the last character in the second line.

- 4 Use the offset value in the `lrs_save_param_ex` function in order to correlate the value of the process ID. For more information, see "Correlating Statements" in *Volume I-Using VuGen*.

Setting the Display Format

You can specify how VuGen will display the buffer data in the viewer (F7) window. The `conv_frm.dat` file in the `lrun/dat` directory contains the following display parameters:

- ▶ **LongBufferFormat.** The format used to display five or more characters. Use `nn` for offset, `XX` for the hex data, and `aa` for ASCII data.
- ▶ **LongBufferHeader.** A header to precede each buffer in Long buffer format.
- ▶ **LongBufferFooter.** A footer to follow each buffer in Long buffer format.
- ▶ **ShortBufferFormat.** The format used to display four characters or less. You can use standard escape sequences and conversion characters.

The supported escape sequence characters are:

<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:

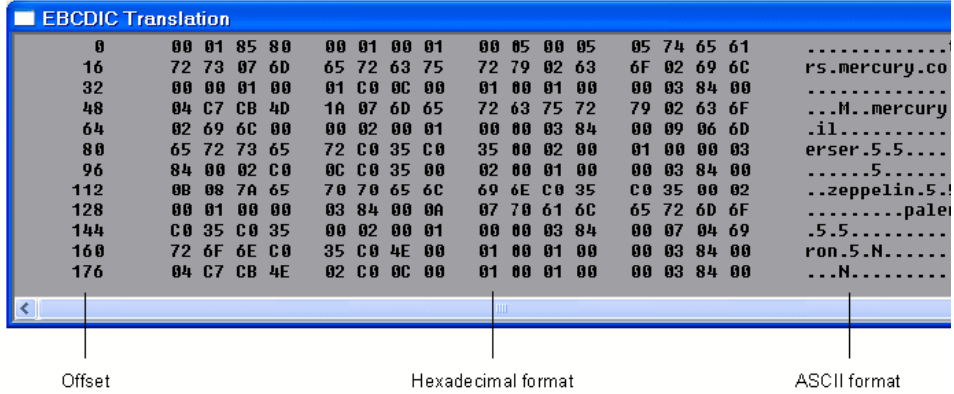
%a	ASCII representation
%BX	Big Endian (Network Order) Hex
%BO	Big Endian (Network Order) Octal
%BD	Big Endian (Network Order) Decimal
%LX	Little Endian Hex
%LO	Little Endian Octal
%LD	Little Endian Decimal

- **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.

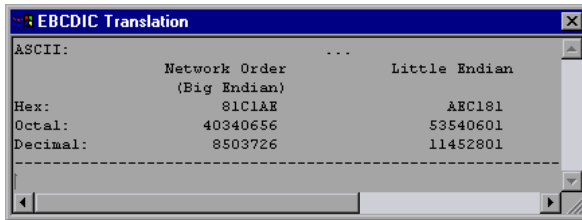
In the default settings, long and short formats are set, and a dot is specified for non-printable characters.

```
[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:



Debugging Tips

VuGen offers several means which allow you to debug your script. You can view the various output logs and windows for detailed messages issued during execution.

Specifically for Windows Sockets Vuser scripts, VuGen provides additional information about buffer mismatches. A buffer mismatch indicates a variation in the received buffer size (generated during replay) and the expected buffer (generated during record). However, if the received and expected buffer are the same size, even though the contents are different, a mismatch message is not issued. This information can help you locate a problem within your system, or with your Vuser script.

You can view the buffer mismatch information in the Execution log. Select **View > Output** to display the Execution log if it is not visible.

Note that a buffer mismatch may not always indicate a problem. For example, if a buffer contains insignificant data such as previous login times, this type of mismatch can be ignored.

```
Mismatch (expected 54 bytes, 58 bytes actually received)
The expected buffer is:
=====
\r\n Last login: Wed Sep 2 10:30:18 from acme.hplab.c\r\n
=====
The received buffer is:
=====
\r\n Last login: Thu Sep 10 11:19:50 from dolphin.hplab.c\r\n
```

However, if there is a very large discrepancy between the size of the Expected and Received buffers, this could indicate a problem with your system. Check the data in the corresponding buffer for discrepancies.

In order for you to determine whether or not the mismatch is significant, you must thoroughly understand your application.

Manually Correlating WinSock Scripts

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

You can manually correlate a Vuser script using the correlation functions that save the dynamic values to a parameter. The **lrs_save_param** and **lrs_save_param_ex** functions let you save data to a parameter based on the offset of the data in the received buffer. An advanced correlation function **lrs_save_searched_string** lets you designate the data by specifying its boundaries and indicating which occurrence of the matched pattern to save to a parameter. The following example describes correlation using **lrs_save_param_ex**. For information about using other correlation functions, see the *Online Function Reference*.

To correlate the WinSock Vuser statements:

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

```
lrs_save_param_ex (socket, type, buffer, offset, length, encoding, parameter);
```

- 2 Reference the parameter.

View the buffer contents by selecting the **data.ws** file in the Data Files box of the main VuGen window. Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in angle brackets (< >).

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

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

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

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

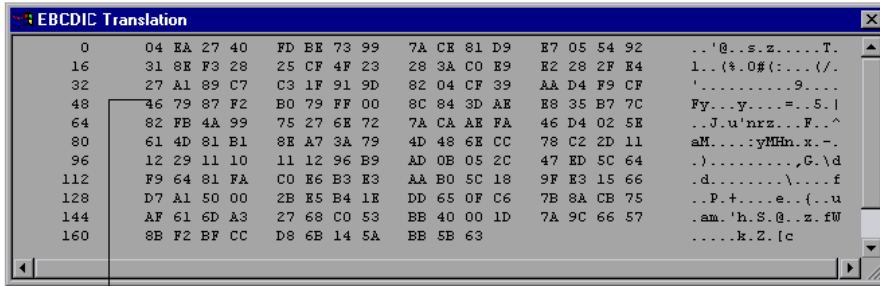
- 3 In the `data.ws` file, determine the buffer in which the data was received, `buf47`.

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

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

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

- Determine the offset and length of the data string to save. Highlight the entire buffer and press F7. The offset of the **PID** is 11 and its length is 5 bytes. For additional information about displaying the data, see "Understanding the Data File Format" on page 403.



Offset of first character in line

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

```
lrs_receive("socket1", "buf79", LrsLastArg);
lrs_save_param("socket1", "user", buf47, 11, 5, ascii, param1);
lr_output_message ("param1: %s", lr_eval_string("<param1>"));
lr_think_time(10);
lrs_send("socket1", "buf80", LrsLastArg);
```

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

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

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

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


26

Programming a Script in the VuGen Editor

In addition to recording a session, you can create a custom Vuser script. You can use both Vuser API functions and standard C, Java, VB, VBScript, or Javascript code.

This chapter includes:

- About Creating Custom Vuser Scripts on page 418
- C Vusers on page 419
- Using the Workflow Wizard for C Vuser Scripts on page 420
- Java Vusers on page 422
- VB Vusers on page 423
- VBScript Vusers on page 425
- JavaScript Vusers on page 426

About Creating Custom Vuser Scripts

VuGen allows you to program your own functions into the script, instead of recording an actual session. You can use the Vuser API or standard programming functions. Vuser API functions allow you to gather information about Vusers. For example, you can use Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test or monitoring.

This chapter describes how to program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes.

You can also develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API function libraries.

To create a customized script, you first create a skeleton script. The skeleton script contains the three primary sections of a script: **init**, **actions**, and **end**. These sections are empty and you manually insert functions into them.

You can create empty scripts for the following programming languages:

- C
- Java
- Visual Basic
- VBScript
- JavaScript

Note: When working with JavaScript and VBScript Vusers, the COM objects that you use within your script must be fully automation compliant. This makes it possible for one application to manipulate objects in another application, or to expose objects so that they may be manipulated.

C Vusers

In C Vuser Scripts, you can place any C code that conforms with the standard ANSI conventions. To create an empty C Vuser script, select C Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty script:

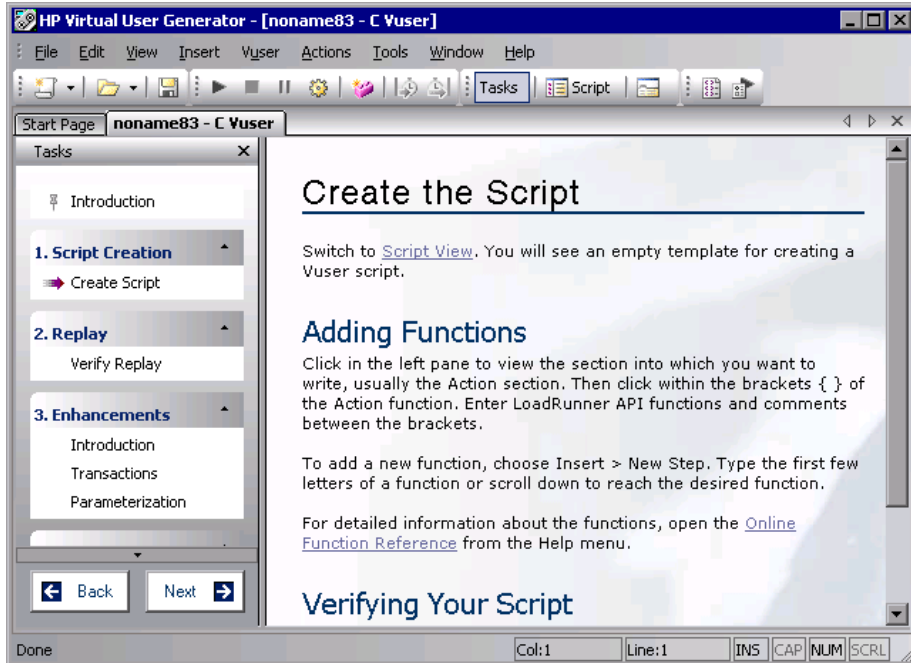
```
Action1()
{
    return 0;
}
```

You can use C Vuser functions in all of Vuser script types that use C functions.

You can also see the *Online Function Reference* (**Help > Function Reference**) for a C reference with syntax and examples of commonly used C functions.

Using the Workflow Wizard for C Vuser Scripts

The Workflow Wizard guides you through the steps of creating a script. By clicking on a link in the Tasks pane, you can read about the steps in creating a script, and view information about your replay. Use the **Back** and **Next** buttons to navigate between screens.



If you do not see the Workflow Wizard, make sure that the Tasks pane is open. (You show and hide the Task pane using the **Tasks** button on the toolbar). Then click the first link, **Introduction**.

See Chapter 4, "Viewing the VuGen Workflow" in *Volume I-Using VuGen* for more information about the wizard.

Create the Script

The Create Script window contains several guidelines for creating a Web Services script.

► **Adding Functions.** describes how and where to enter the functions.

- **Verifying Your Script.** describes how to verify your script after adding functions.

Guidelines for Using C Functions

All standard ANSI-C conventions apply to C Vuser scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other C programs. You declare and define variables using ANSI C conventions.

The C interpreter that is used to run Vuser scripts accepts the standard ANSI C language. It does not support any Microsoft extensions to ANSI C.

Before you add any C functions to a Vuser script, note the following limitations:

- A Vuser script cannot pass the address of one of its functions as a callback to a library function.
- The **stdargs**, **longjmp**, and **alloca** functions are not supported in Vuser scripts.
- Vuser scripts do not support structure arguments or return types. Pointers to structures are supported.
- In Vuser scripts, string literals are read-only. Any attempt to write to a string literal generates an access violation.
- C Functions that do not return int, must be casted. For example, `extern char * strtok();`

Calling libc Functions

In a Vuser script, you can call **libc** functions. However, since the interpreter that is used to run Vuser scripts does not support any Microsoft extensions to ANSI C, you cannot use Microsoft's include files. You can either write your own prototypes, or ask HP Customer Support to send you ANSI-compatible include files containing prototypes for **libc** functions.

Linking Mode

The C interpreter that is used to run Vuser scripts uses a "lazy" linking mode in the sense that a function need not be defined at the start of a run, as long as the function is defined before it is used. For example:

```
lr_load_dll("mydll.dll");
    myfun(); /* defined in mydll.dll -- can be called directly,
            immediately after myfun.dll is loaded. */
```

Java Vusers

In Java Vuser scripts, you can place any standard Java code. To create an empty Java Vuser script, select Java Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty Java script:

```
import Irapi.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.

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 VBScript Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty VBScript Vuser script:

```
Public Function Actions()

    "TO DO: Place your action code here

    Actions = lr.PASS
End Function
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a VBScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the TO DO comment.

An additional section that is viewable from VuGen, is the **global.vbs** file, which creates the objects for the Vuser API functions and VB Script. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
Set lr = CreateObject("LoadRunner.LrApi")
```

JavaScript Vusers

You can create an empty JavaScript Vuser script, in which to place JavaScript code. This script type lets you incorporate your existing JavaScript application into VuGen. To create an empty JavaScript Vuser script, select JavaScript Vuser from the Custom category, in the New Virtual User dialog box.

```
function Actions()
{
    /*TO DO: Place your business process/action code here

    return(lr.PASS);
}
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a JavaScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the TO DO comment.

An additional section that is viewable from VuGen, is the **global.js** file, which creates the objects for the Vuser API functions and the Javascript. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
var lr = new ActiveXObject("LoadRunner.LrApi")
```

27

Programming Java Scripts

VuGen supports Java type users on a protocol level. This chapter explains how to create a Java Vuser script by programming. For information on creating a Java Vuser script through recording, see the chapters describing the Java protocol.

This chapter includes:

- About Programming Java Scripts on page 428
- Creating a Java Vuser on page 429
- Editing a Java Vuser Script on page 429
- Java Vuser API Functions on page 431
- Working with Java Vuser Functions on page 433
- Setting your Java Environment on page 439
- Running Java Vuser Scripts on page 439
- Compiling and Running a Script as Part of a Package on page 440
- Programming Tips on page 441

About Programming Java Scripts

To prepare Vuser scripts using Java code, use the **Java** type Vusers. This Vuser type supports Java on a protocol level. The Vuser script is compiled by a Java compiler and supports all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes `"/"`.

Chapter 14, "Java Protocols - Recording" explains how to create a script through recording using the **Java Record Replay** Vuser. To prepare a Java coded script through programming, see the following sections.

The first step in creating a Java compatible Vuser script, is to create a new Vuser script template of the type **Java Vuser**. Then, you program or paste the desired Java code into the script template. You can add Java Vuser functions to enhance the script and parameterize the arguments to use different values during iterations.

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

After you prepare a script, run it as a standalone test from VuGen. A Java compiler (Sun's `javac`), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Creating a Java Vuser

The first step in creating a Java-compatible Vuser script is creating a Java Vuser template.

To create a Java Vuser script:

- 1** Open VuGen.
- 2** Select **File > New** or click the **New** button. The New Virtual User dialog box opens.
- 3** Select **Custom > Java Vuser** from the Select Vuser type list, and click **OK**. VuGen displays a blank Java Vuser script.
- 4** Click the **Actions** section in the left frame to display the **Actions** class.

Editing a Java Vuser Script

After generating an empty template, you can insert the desired Java code. When working with this type of Vuser script, you place all your code in the Actions class. To view the Actions class, click **Actions** in the left pane. VuGen displays its contents in the right pane.

```
import 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...
<code>init</code>	a login to a server	the Vuser is initialized (loaded)
<code>action</code>	client activity	the Vuser is in "Running" status
<code>end</code>	a log off procedure	the Vuser finishes or is stopped

Init Method

Place all the login procedures and one-time configuration settings in the `init` method. The `init` method is only executed once—when the Vuser begins running the script. The following sample `init` method initializes an applet.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import Irapi.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 Vuser actions in the `action` method. The `action` method is executed according to the number of iterations you set in the runtime settings. For more information on the iteration settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*. The following sample `action` method retrieves and prints the Vuser ID.

```
public int action() {
    lr.message("vuser: " + lr.get_vuser_id() + " xxx");
    return 0;
}
```

End Method

In the **end** method, place the code you want the Vuser to execute at the end of the script, such as logging off from a server, cleaning up the environment, and so forth.

The end method is only executed once—when the Vuser finishes running the script. In the following example, the end method closes and prints the end message to the execution log.

```
public int end() {  
    lr.message("End");  
    return 0;  
}
```

Java Vuser API Functions

VuGen provides a specific Java API for Java Vuser scripts. These functions are all static methods of the `lrapi.lr` class.

The Java API functions are classified into several categories: Transaction, Command Line Parsing, Informational, String, Message, and Run-Time functions.

For further information about each of these functions, see the *Online Function Reference* (**Help > Function Reference**). Note that when you create a new Java Vuser script, the `import lrapi.*` is already inserted into the script.

To use additional Java classes, import them at the beginning of the script as shown below.

Remember to add the classes directory or relevant jar file to the classpath. Make sure that the additional classes are thread-safe and scalable.

```
import java.io.*;
import Irapi.*;

public class Actions
{
  ...
}
```


Working with Java Vuser Functions

You can use Java Vuser functions to enhance your scripts by:

- Inserting Transactions
- Inserting Rendezvous Points
- Obtaining Vuser Information
- Issuing Output Messages
- Emulating User Think Time
- Handling Command Line Arguments

Inserting Transactions

You define transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified requests. These requests can be short or complex tasks. When working with LoadRunner, you can analyze the performance per transaction during and after the scenario run, using online monitor and graphs.

You can also specify a transaction status: lr.PASS or lr.FAIL. You can let the Vuser automatically determine if the transaction was successful, or you can incorporate it into a conditional loop. For example, in your code you can check for a specific return code. If the code is correct, you issue a lr.PASS status. If the code is wrong, you issue an lr.FAIL status.

To mark a transaction:

- 1** Insert **lr.start_transaction** into the script, at the point where you want to begin measuring the timing of a task.
- 2** Insert **lr.end_transaction** into the script, at the point where you want to stop measuring the task. Use the transaction name as it appears in the **lr.start_transaction** function.

3 Specify the desired status for the transaction: lr.PASS or lr.FAIL.

```
public int action() {
    for(int i=0;i<10;i++)
    {
        lr.message("action()"+i);
        lr.start_transaction("trans1");
        lr.think_time(2);
        lr.end_transaction("trans1",lr.PASS);
    }
    return 0;
}
```

Inserting Rendezvous Points

The following section does not apply to the HP Business Availability Center.

To emulate heavy user load on your client/server system, you synchronize Vusers to perform a task at exactly the same moment by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it is held by the Controller until all Vusers participating in the rendezvous arrive.

You designate the meeting place by inserting a rendezvous function into your Vuser script.

To insert a rendezvous point:

- Insert an lr.rendezvous function into the script, at the point where you want the Vusers to perform a rendezvous.

```
public int action() {
    for(int i=0;i<10;i++)
    {
        lr.rendezvous("rendz1");
        lr.message("action()"+i);
        lr.think_time(2);
    }
    return 0;
}
```

Obtaining Vuser Information

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

lr.get_attrib_string	Returns a string containing command line argument values or runtime information such as the Vuser ID or the load generator name.
lr.get_group_name	Returns the name of the Vuser's group.
lr.get_host_name	Returns the name of the load generator executing the Vuser script.
lr.get_master_host_name	Returns the name of the machine running the LoadRunner Controller or Business Process Monitor.
lr.get_scenario_id	Returns the ID of the current scenario. (LoadRunner only)
lr.get_vuser_id	Returns the ID of the current Vuser. (LoadRunner only)

In the following example, the `lr.get_host_name` function retrieves the name of the computer on which the Vuser is running.

```
String my_host = lr.get_host_name();
```

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

Issuing Output Messages

When you run a scenario, the Controller Output window displays information about script execution. You can include statements in a Vuser script to send error and notification messages to the Controller. The Controller displays these messages in the Output window. For example, you could insert a message that displays the current state of the client application. You can also save these messages to a file.

Note: Do not send messages from within a transaction. Doing so lengthens the transaction execution time and may skew the actual transaction results.

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

lr.debug_message	Sends a debug message to the Output window.
lr.log_message	Sends a message to the Vuser log file.
lr.message	Sends a message to a the Output window.
lr.output_message	Sends a message to the log file and Output window with location information.

In the following example, **lr.message** sends a message to the output indicating the loop number:

```
for(int i=0;i<10;i++)
{
    lr.message("action()"+i);
    lr.think_time(2);
}
```

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

You can instruct the Vusers to redirect the Java standard output and standard error streams to VuGen's Execution log. This is especially helpful when you need to paste existing Java code or use ready-made classes containing **System.out** and **System.err** calls in your Vuser scripts. In the execution log, standard output messages are colored blue, while standard errors are shown in red.

The following example shows how to redirect specific messages to the standard output and standard error using `lr.enable_redirection`:

```
lr.enable_redirection(true);

System.out.println("This is an informatory message..."); // Redirected
System.err.println("This is an error message..."); // Redirected

lr.enable_redirection(false);

System.out.println("This is an informatory message..."); // Not redirected
System.err.println("This is an error message..."); // Not redirected
```

Note: When you set `lr.enable_redirection` to **true**, it overrides all previous redirections. To restore the former redirections, set this function to **false**.

For additional information about this function, see the *Online Function Reference* (**Help > Function Reference**).

Emulating User Think Time

The time that a user waits between performing successive actions is known as the think time. Vusers use the `lr.think_time` function to emulate user think time. In the following example, the Vuser waits two seconds between loops:

```
for(int i=0;i<10;i++)
{
    lr.message("action()"+i);
    lr.think_time(2);
}
```

You can use the think time settings as they appear in the script, or a factor of these values. To configure how Vusers handle think time functions, open the runtime settings dialog box. For more information, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

For more information about the `lr.think_time` function, see the *Online Function Reference (Help > Function Reference)*.

Handling Command Line Arguments

You can pass values to a Vuser script at runtime by specifying command line arguments when you run the script. You insert command line options after the script path and filename in the Controller or Business Process Monitor. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

lr.get_attrib_double	Retrieves double precision floating point type arguments
lr.get_attrib_long	Retrieves long integer type arguments
lr.get_attrib_string	Retrieves character strings

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

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

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

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

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

For more information on the command line parsing functions, see the *Online Function Reference (Help > Function Reference)*. For more information on how to insert the command line options, see the *LoadRunner Controller, Performance Center, or HP Business Availability Center* documentation.

Setting your Java Environment

Before running your Java Vuser script, make sure that the environment variables, `PATH` and `CLASSPATH`, are properly set on all machines running Vusers:

- ▶ To compile and replay the scripts, you must have complete JDK installation, either version 1.1 or 1.2, or 1.3. The installation of the JRE alone is not sufficient. It is preferable not to have more than one JDK or JRE installation on a machine. If possible, uninstall all unnecessary versions.
- ▶ The `PATH` environment variable must contain an entry for `JDK/bin`.
- ▶ For JDK 1.1.x, the `CLASSPATH` environment variable must include the `classes.zip` path, (`JDK/lib` subdirectory) and all of the VuGen classes (`classes` subdirectory).
- ▶ All classes used by the Java Vuser must be in the classpath—either set in the machine's `CLASSPATH` environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

Running Java Vuser Scripts

Java Vuser scripts differ from C Vuser scripts in that they are first compiled and then executed; C Vuser scripts are interpreted. VuGen locates the `javac` compiler from within the JDK installation and compiles the Java code inside the script. This stage is indicated by the **Compiling...** status message in the bottom of the VuGen window. If errors occur during compilation, they are listed in the execution log. To go to the code in your script that caused the error, double-click on the error message containing the line number of the error. Fix the error and run the script again.

If the compilation succeeds, the status message **Compiling...** changes to **Running...** and VuGen begins to execute the script. When you run the script again, VuGen runs the script without recompiling it, provided that no changes were made to the script. To debug your script further, you can use breakpoints and animated run type execution using the step option.

Note: If you are making calls to JNDI extensions within your script, you may encounter problems trying to run your Vusers as **threads**. This happens because JNDI requires each thread to have its own context class loader. In order to run as threads, instruct each Vuser to run with its own context class loader, by adding the following line to the beginning of the **init** section:

```
DummyClassLoader.setContextClassLoader();
```

Compiling and Running a Script as Part of a Package

When creating a Java Vuser script, you may need to use methods in other classes in which the class or method is protected. If you try to compile this type of script, you will receive errors in the compilation stage indicating that the methods are inaccessible. To make sure that your script can access these methods, insert the package name containing these methods at the top of the script, just as you would do in a standard Java program—`<package_name>`. In the following example, the script defines the `just.do.it` package which consists of a path:

```
package my.test;

import Irapl.*;
public class Actions
{
    :
}
```

In the above example, VuGen automatically creates the **my/test** directory hierarchy under the Vuser directory, and copies the **Actions.java** file to **my/test/Actions.java**, allowing it to compile with the relevant package. Note that the package statement must be the first line in the script, similar to Java (excluding comments).

Programming Tips

When programming a Java Vuser script, you can paste ready-made code segments into scripts or import ready-made classes in order to invoke their methods. If Vusers need to run as threads under the Controller (for scalability reasons), you need to make sure that all of the imported code is thread-safe.

Thread-safety is often difficult to detect. A Java Vuser may run flawlessly under VuGen and under the Controller with a limited number of Vusers. Problems occur with a large number of users. Code that is not thread-safe is usually the result of static class member usage as shown in the following example:

```
import 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 "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

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;
}
```

28

COM Protocol

Many Windows applications use COM-based functions either directly, or through library calls. You can use VuGen to record a script that emulates a COM-based client accessing a COM server. The resulting script is called a COM Vuser script. You can also create COM Vuser scripts by using a Visual Basic add-in. For more information about the Visual Basic add-in, see the Virtual User Guide appendix.

Chapter 29, "COM - Understanding and Correlating," explains how COM Vuser scripts operate.

This chapter includes:

- About Recording COM Vuser Scripts on page 444
- COM Overview on page 444
- Getting Started with COM Vusers on page 446
- Selecting COM Objects to Record on page 447
- Setting COM Recording Options on page 450

About Recording COM Vuser Scripts

When you record COM client applications, VuGen generates functions that describe COM client-server activity. The recorded script contains interface declarations, API calls and instance calls to methods. Each COM function begins with an `Irc` prefix.

You can view and edit the recorded script from the VuGen's main window. The COM API/method calls that were recorded during the session are displayed in the window, allowing you to visually track application COM/DCOM calls.

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. See Chapter 19, "Setting Script Generation Preferences" in *Volume I-Using VuGen* for more information.

COM Overview

This section provides an outline of COM technology. This should be enough to get you started with COM Vuser scripts. See Microsoft Developer's Network (MSDN) and other documentation for further details.

COM (Component Object Model) is a technology for developing reusable software components ("plug-ins"). DCOM (Distributed COM) allows use of COM components on remote computers. Microsoft transaction servers (MTS), Visual Basic and Explorer all use COM/DCOM technology. Thus, the application you are testing may use COM technology indirectly, even though you don't know it. You will probably have to include some, but certainly not all, of the COM calls made by your application in the Vuser script.

Objects, Interfaces and Type Libraries

COM objects are binary code modules. Each COM object implements one or more interfaces that allow client programs to communicate with it. You need to know about these interfaces in order to follow the COM calls in the Vuser scripts. Type libraries, used as a reference for accessing COM interface methods and parameters, contain descriptions of COM objects and interfaces. Each COM class, interface, and type library is identified by a Global Unique Identifier (GUID).

COM Interfaces

A COM interface provides a grouped collection of related methods. For example, a **Clock** object may have **Clock**, **Alarm** and **Timer** interfaces. Each interface has one or more methods. For example the **Alarm** interface may have **AlarmOn** and **AlarmOff** methods.

An interface may also have one or more properties. Sometimes, the same function may be performed by calling a method or by setting or getting the value of a property. For example, you can set the **Alarm Status** property to **On** or call the **AlarmOn** method.

A COM object may support many interfaces. The **IUnknown** interface is implemented by all components and is used to find out about other interfaces. Many components also implement the **IDispatch** interface, which exposes all other interfaces and methods of the object, allowing implementation of COM automation in scripting languages.

COM Class Context and Location Transparency

COM objects can run on the same machine as the client application, or on a remote server. COM objects that an application creates may be in a local library, a local process or a remote machine ("Remote Object Proxy"). The location of the COM object, known as the "Context," can be transparent to the application. Most users apply the Vusers to check the load on remote servers. Therefore, objects accessed by Remote Object Proxy are usually the most relevant for these tests.

COM Data Types

COM also provides several special data types, including safe arrays, BSTR strings and variants. You may need to use these data types for debugging, parameterization and similar tasks.

Getting Started with COM Vusers

This section describes the process of developing COM Vuser scripts.

To develop a COM Vuser script:

1 Record the basic script using VuGen.

Start VuGen and create a new Vuser script. Specify COM as the type of Vuser. Select an application to record and set the recording options. To set the script related recording options, see "Setting Script Generation Preferences" in *Volume I-Using VuGen*. To set the COM specific options and filters, see the "Setting COM Recording Options" on page 450. Record typical operations using your application.

For details about recording, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Refine the Object Filter.

Use the log file that was generated to refine your choice of objects to be recorded in the filter. See the following section, "Selecting COM Objects to Record", for details.

3 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Selecting COM Objects to Record

The application you are testing may use a great many COM objects. Only a few may actually create load and may be important for the load test. Thus, before you record a COM application, you should select the objects you want to record for the load test. VuGen allows you to browse for objects from type libraries that it can read on the local machine and on other computers in the network.

Deciding Which Objects to Use

There are several ways to decide which COM objects should be included in the test. Try to determine which remote objects are used by the software. If you are unsure which objects to use, try using the default filter. The Environments branch of the filter includes calls to three sets of objects (ADO, RDS and Remote) that are likely to generate load on remote servers.

You can also check the actual calls to refine the filter. After you have recorded the test, you can save the file and look in the **data** directory that VuGen creates for a file named **lrc_debug_list_<nnn>.log**, where **nnn** is the process number. This log file contains a listing of each COM object that was called by the recorded application, regardless of whether or not the recording filter included that object. Only calls that generate load on the server should be included for recording.

For example, the following is a local COM of the Visual Basic library:

```
Class JetES {039EA4C0-E696-11D0-878A-00A0C91EC756}
was loaded from type library "JET Expression Service Type Library"
({2358C810-62BA-11D1-B3DB-00600832C573} ver 4.0)
```

It should not be added since it does not generate load on the server.

Likewise, since the OLE DB and Microsoft Windows Common Controls are local objects, the following are examples of classes and libraries that are not going to place any load on the server and should not be recorded:

```
Class DataLinks {2206CDB2-19C1-11D1-89E0-00C04FD7A829}
was loaded from type library "Microsoft OLE DB Service Component 1.0 Type Library"
({2206CEB0-19C1-11D1-89E0-00C04FD7A829} ver 1.0)

Class DataObject {2334D2B2-713E-11CF-8AE5-00AA00C00905}
was loaded from type library "Microsoft Windows Common Controls 6.0 (SP3)"
({831FDD16-0C5C-11D2-A9FC-0000F8754DA1} ver 2.0)
```

However, for example, a listing such as the following indicates a class that should be recorded since it does generate load on the server:

```
Class Order {B4CC7A90-1067-11D4-9939-00105ACECF9A}
was loaded from type library "FRS"
({B4CC7A8C-1067-11D4-9939-00105ACECF9A} ver 1.0)
```

Calls to classes of the **FRS** library, used for instance in the **flight_sample** that is installed with VuGen, use server capacity and should be recorded.

If a COM object itself calls other COM objects, all the calls will be listed in the type information log file. For example, every time the application calls an **FRS** class function, the **FRS** library calls the **ActiveX Data Object (ADO)** library. If several functions in such a chain are listed in a filter, VuGen records only the first call that initiates the chain. If you selected both **FRS** and **ADO** calls, only the **FRS** calls will be recorded.

On the other hand, if you select only the **ADO** library in the filter, then calls to the **ADO** library will be recorded. It is often easier to record the call to the first remote object in the chain. In some cases, however, an application may use methods from several different COM objects. If all of them use a single object that puts a load on the server, you could only record the final common object.

Which Objects Can Be Selected

VuGen can only record objects if it can read their type libraries. If the type libraries were not installed in the system or VuGen cannot find them, the COM objects will not be listed in the Recording Options dialog box. If they are used by your application, VuGen will not be able to identify these objects and will identify them as **INoTypeInfo** in the files.

Which Interfaces Can Be Excluded

For each object, the Recording Options dialog box will show you all interfaces that are listed in the Type Library, and allow you to specify inclusion or exclusion of each one. However, **ADO**, **RDS** and **Remote Objects** can be included in the filter as a group. The filter will not show the individual objects of those environments or their interfaces. Objects that you included from type libraries may also have interfaces that are not listed in the type library and therefore not shown in the Recording Options dialog. After generating a VuGen script, you can identify these interfaces in the script and get their GUID numbers from the interfaces.h file that VuGen generates. Using this information, you can exclude the interfaces as explained below.

Setting COM Recording Options

Use the COM Recording Options dialog box to set the filtering and COM scripting options. You use the online browser to locate type libraries in the registry, file system, or the Microsoft Transaction Server (MTS).

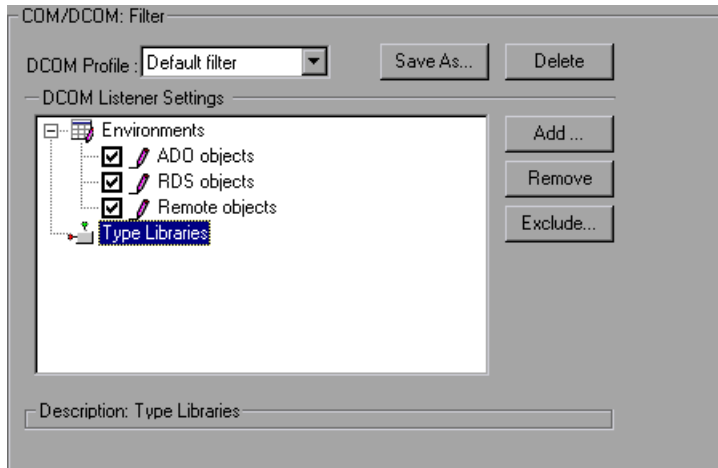
For more information, see:

- Filtering Objects
- Setting the Filter
- Setting COM Scripting Options

Filtering Objects

The Filter options let you indicate which COM objects should be recorded by VuGen. You can select objects from within environments and libraries.

The Filter options set a default filter or create alternate filters. You can filter a recording session by environment and type libraries.



DCOM Profile

- ▶ **Default Filter.** The filter to be used as the default when recording a COM Vuser script.
- ▶ **New Filter.** A clean filter based on the default environment settings. Note that you must specify a name for this filter before you can record with its settings.

DCOM Listener Settings

The DCOM Listener Settings display a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, a dialog box opens asking you if you also want to exclude the interface in all classes that implement it this interface.

Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

- ▶ **Environment.** The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record.
- ▶ **Type Libraries.** A type library **.tlb** or **.dll** file, that represents the COM object to record. All COM objects have a type library that represents them. You can select a type library from the Registry, Microsoft Transaction Server, or file system.

Type Libraries. In the lower section of the dialog box, VuGen displays the following information for each type library.

- ▶ **TypLib.** The name of the type library (tlb file).
- ▶ **Path.** The path of the type library.
- ▶ **Guid.** The Global Unique Identifier of the type library.

Setting the Filter

This section describes how to set the filters.

To select which COM objects to record:

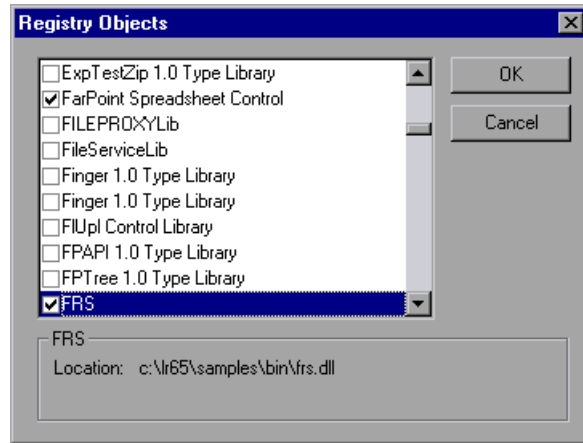
- 1** Select **Tools > Recording Options** from the main menu or click **Options** in the Start Recording dialog box. A dialog box opens displaying the Recording Options tree. Select the **COM/DCOM:Filter** node.

Expand the Environments sub-tree, to display the **ADO, RDS** and **Remote objects** listings. The Filter also includes a **Type Libraries** tree that is initially empty. You can add Type Libraries as described in the steps below.

By default, all Environments are selected and calls to any of their objects are included in the filter. Clear the check box adjacent to **ADO, RDS** or **Remote objects** to exclude them from the filter.

- 2** Click **Add** to add another COM type library, and select a source to browse: registry, file system, or MTS, as described below.

- 3 Select **Browse Registry** to display a list of type libraries found in the registry of the local computer.



Select the check box next to the desired library or libraries and click **OK**.

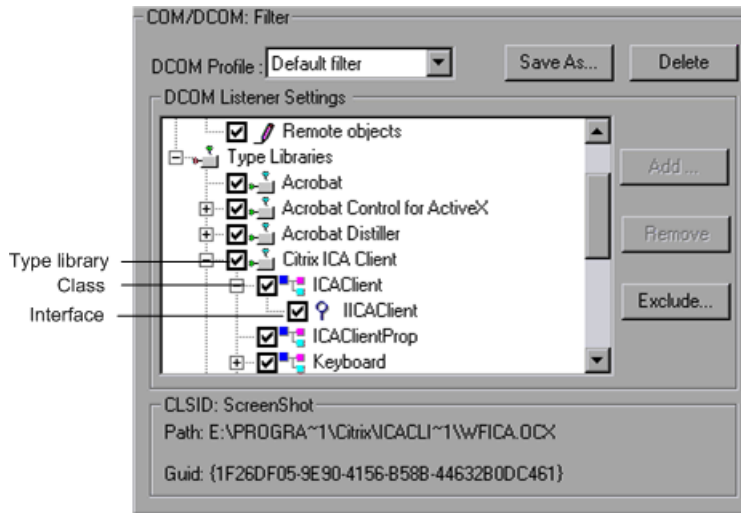
- 4 To add a type library from the file system, click **Add** and select **Browse file system**.

Select the desired file and click **OK**.

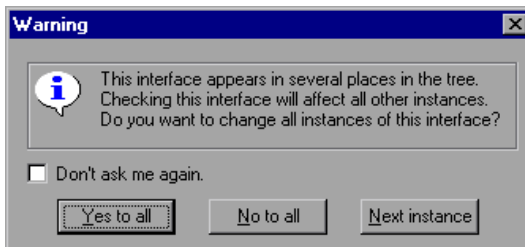
- 5 Once the type library appears in the list of Type Libraries, you can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

When you clear a check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

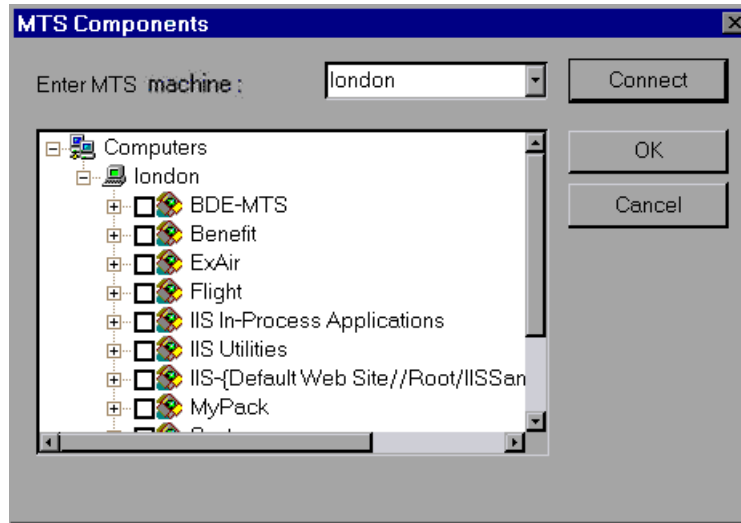


- An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the following warning:



If you check **Don't ask me again** and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click **Yes to all** to change the status of all instances of this interface for all other classes, click **No to all** to leave the status of all other instances unchanged. Click **Next Instance** to view the next class that uses this interface.

- 7 To add a component from a Microsoft Transaction Server, click **Add** and select **Browse MTS**. The MTS Components dialog box prompts you to enter the name of the MTS server.

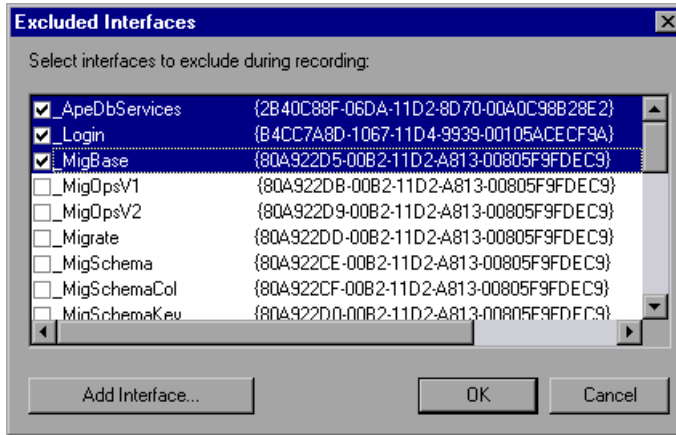


Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine.

Select one or more packages of MTS components from the list of available packages and click **Add**. Once the package appears in the list of Type Libraries, you can select specific components from the package.

- 8 In addition to disabling and enabling recording of interfaces in the tree display, you can also click **Exclude** in the Recording Options dialog to include or exclude interfaces in the filter, whatever their origin.

Note that you can also exclude classes and interfaces by clearing the check box adjacent to the item, inside the type library tree hierarchy.



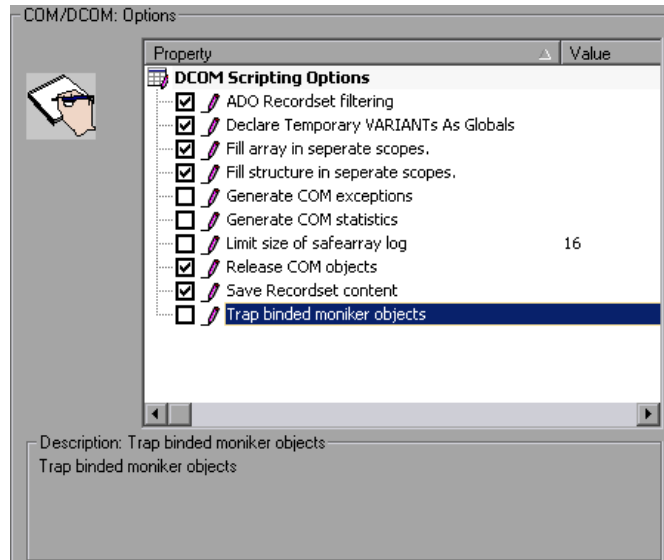
The checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click **Add Interface...** in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the **Add Interface...** feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter.

- 9 When you have finished modifying your filter, click **OK** to save it and close the dialog box. Click **Save As** to save a New filter, or to save an existing filter under a new name. You can select saved filters in subsequent recordings. Default settings are given initially in the **Default filter**.

Setting COM Scripting Options

You can set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.



- **ADO Recordset filtering.** Condense multiple recordset operations into a single-line fetch statement (enabled by default).
- **Declare Temporary VARIANTS as Globals.** Define temporary VARIANT types as Globals, not as local variables (enabled by default).
- **Fill array in separate scopes.** Fill in each array in a separate scope (enabled by default).
- **Fill structure in separate scopes.** Fill in each structure in a separate scope (enabled by default).
- **Generate COM exceptions.** Generate COM functions and methods that raised exceptions during recording (disabled by default).
- **Generate COM statistics.** Generate recording time performance statistics and summary information (disabled by default).

- ▶ **Limit size of SafeArray log.** Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).
- ▶ **Release COM Objects.** Record the releasing of COM objects when they are no longer in use (enabled by default).
- ▶ **Save Recordset content.** Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).
- ▶ **Trap binded moniker objects.** Trap all of the bound moniker objects (disabled by default).

To set COM/DCOM options:

- 1** Select **Tools > Recording Options** from the main menu or click **Options** in the Start Recording dialog box. VuGen opens the Recording Options tree. Select the **COM/DCOM:Options** node.
- 2** Enable the desired options by clicking the check boxes adjacent to them.
- 3** Click **OK** to save your settings and exit.

29

COM - Understanding and Correlating

This chapter provides details about the scripts VuGen generates for COM client communications, including an explanation of the function calls and examples. For basic information about getting started with COM Vuser scripts, see Chapter 28, "COM Protocol."

This chapter includes:

- About COM Vuser Scripts on page 459
- Understanding VuGen COM Script Structure on page 460
- Examining Sample VuGen COM Scripts on page 462
- Scanning a Script for Correlations on page 468
- Correlating a Known Value on page 470

About COM Vuser Scripts

For each COM Vuser script, VuGen creates the following:

- Interface pointer and other variable declarations in file interfaces.h
- Function calls that you can record in the vuser_init, actions or vuser_end sections.
- A user.h file containing the translation of the Vuser script into low level calls

When you record COM client communications, VuGen creates a script with calls to COM API functions and interface methods. In addition, you can manually program COM type conversion functions.

After you record the script, you can view any of these files by selecting them from the tree on the left-hand side of the VuGen screen.

Each VuGen COM function has an **Irc** prefix, such as **Irc_CoCreateInstance** or **Irc_long**. The Irc functions are classified into several categories: Creating Instances, IDispatch Interface Calls, Type Conversion from String, Assignment to Variants, Create New Variants, Parameterization, Extracting From Variants, Array Types, ADO Recordset, Byte Array, VB Collection Support, and Debug functions.

For syntax and examples of the Irc functions, see the *Online Function Reference* (**Help > Function Reference**).

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see "Setting Script Generation Preferences" in *Volume I-Using VuGen*.

Understanding VuGen COM Script Structure

VuGen COM scripts are structured in a special way to meet the needs of COM interfaces.

Interface Methods

Calls to interface methods have the following names and syntax conventions:

```
Irc_<interface name>_<method name>(instance,...);
```

Note that the **instance** is always the first parameter passed.

The vendors of the respective COM components usually supply documentation for the interface functions.

Interface Pointers

The interface header file defines the interface pointers, as well as other variables, that can be used in the script. Each interface has an Interface ID (IID) which uniquely identifies the interface.

The format of the interface definition is:

```
<interface type>*<interface name> = 0; ///{<IID of the interface type>}"
```

In the following example, the interface type is IDispatch, the name of the interface instance is IDispatch_0, and the IID of IDispatch type is the long number string:

```
IDispatch* IDispatch_0= 0;///{00020400-0000-0000-C000-000000000046}"
```

Vuser Script Statements

The COM Vuser script consist of code that creates object instances, retrieves interface pointers and calls the interface methods. Each user action may generate one or more COM calls. Each COM call is coded by VuGen as a group of statements. Each such group is contained in a separate scope enclosed in braces. Several different statements prepare for the main call by assigning values and performing type conversions. For example, the group of calls needed to create an object may look like this:

```
{
  GUID pClsid = lrc_GUID("student.student.1");
  IUnknown * pUnkOuter = (IUnknown*)NULL;
  unsigned long dwClsContext = lrc_ulong("7");
  GUID riid = IID_IUnknown;
  lrc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid, (void**)&IUnknown_0,
  CHECK_HRES);
}
```

Error Checking

Each COM method or API call returns an error value. VuGen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

CHECK_HRES	This value is inserted if the function passed during recording and errors should be checked during replay.
DONT_CHECK_HRES	This value is inserted if the function failed during recording and errors should not be checked during replay.

Examining Sample VuGen COM Scripts

This section shows examples of how VuGen emulates a COM client application.

Note that VuGen displays the results of a query in a grid. You can view up to 200 records by scrolling through the grid. For more information, see "Working with Grids" on page 366

Basic COM Script Operations

The basic operations are:

- ▶ Instantiation of the object
- ▶ Retrieving interface pointers
- ▶ Calling interface methods

Each type of operation is done within a separate scope.

Instantiation of the Object

To use a COM object, the application must first instantiate it and get a pointer to an interface of that object.

VuGen does the following to instantiate an object:

- 1 VuGen calls `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 = Irc_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:

```
Irc_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.

```
Irc_Istudent_get_name(Istudent_0, &pVal, CHECK_HRES);
```

- 3 VuGen then generates a statement for saving the values.

```
//Irc_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 = Irc_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_DispMethod` 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_DispMethod((IDispatch*)IOptional_0, "in_out_optional_args", /*locale*/1024,
&P1, &P2, OPTIONAL_ARGS, "#3", &P3, "var3", &var3, LAST_ARG, CHECK_HRES);
}
```

The different `Irc_Disp` methods that use the **IDispatch** interface are detailed in the *Online Function Reference*.

Type Conversions and Data Extraction

As shown in the above example, many COM parameters are defined as variants. To extract these values, VuGen uses a number of conversion functions, derived from the equivalent COM functions. The full list is given in the *Online Function Reference*. Previously, we showed how the `Irc_DispMethod1` call was used to retrieve an array of name strings:

```
VARIANT retValue = Irc_variant_empty();
retValue = Irc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray", /*locale*/1033,
LAST_ARG, CHECK_HRES);
```

The following example now shows how VuGen gets the strings out of **retValue**, which is a variant that will be read as an array of strings.

First, VuGen extracts the BSTR array from the variant:

```
BstrArray array0 = 0;  
array0 = 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**)

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script and assist you in getting a successful replay. It performs the following steps:

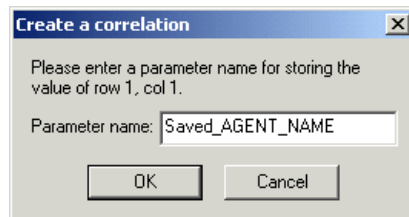
- scans for potential correlations
- insert the appropriate correlation function to save the results to a parameter
- replace the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

To scan and correlate a script detected with automatic correlation:

- 1** Select **View > Output** to display the output tabs at the bottom of the window. Check for errors in the Replay Log tab. Often, these errors can be corrected by correlation.
- 2** Select **Vuser > Scan for Correlations**. VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.
- 3** Correlate the value. In the Correlated Query tab, double-click on the result you want to correlate. This is located on the line of the message where it says
grid column x, row x.
VuGen sends the cursor to the grid location of the value in your script.
- 4** In the grid, select **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.



- 5** Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrc_save_<type>**) which saves the result to a parameter.
- 6** Click **Yes** to confirm the correlation.
- 7** A message box opens asking if you want to search for all occurrences of the value in the script.
Click **No** to replace only the value in the selected statement.
To search and replace additional occurrences click **Yes**.
- 8** A Search and Replace dialog box opens. Confirm any replacements, including your original statement.
- 9** Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. Note that if you cancel the correlation, VuGen also erases the statement created in the previous step.

Correlating a Known Value

If you know which value needs to be correlated, perform the following procedure:

To correlate a specific value:

- 1 Locate the argument you want to correlate (usually in an `Irc_variant_` statement) and select the value without the quotation marks.

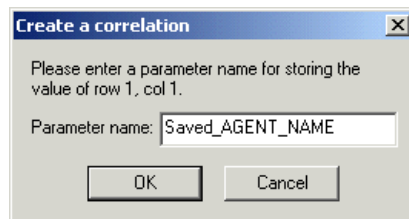
- 2 Select **Vuser > Scan for Correlations (at cursor)**.

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 (`Irc_save_<type>`) which saves the result to a parameter.

```
Irc_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.

30

AJAX (Click and Script) Protocol

VuGen allows you to create scripts that emulate AJAX (Asynchronous JavaScript and XML) enabled applications.

This chapter includes:

- ▶ About Developing AJAX (Click and Script) Vuser Scripts on page 473
- ▶ Recording an AJAX (Click and Script) Session on page 474
- ▶ Understanding AJAX (Click and Script) Scripts on page 474

About Developing AJAX (Click and Script) Vuser Scripts

AJAX (Asynchronous JavaScript and XML) is a technique for creating interactive Web applications. With AJAX, Web pages exchange small packets of data with the server, instead of reloading an entire page. This reduces the amount of time that a user needs to wait when requesting data. It also increases the interactive capabilities and enhances the usability.

Using AJAX, developers can create fast Web pages using Javascript and asynchronous server requests. The requests can originate from user actions, timer events, or other predefined triggers.

AJAX components, also known as AJAX controls, are GUI based controls that use the AJAX technique—they send a request to the server when a trigger occurs.

For example, a popular AJAX control is a **Reorder List** control that lets you drag components to a desired position in a list. VuGen's support for AJAX implementation is based on Microsoft's ASP.NET AJAX Control Toolkit formerly known as Atlas.

The supported frameworks for AJAX functions are:

- ▶ Atlas 1.0.10920.0/ASP.NET AJAX—All controls
- ▶ Scriptaculous 1.8—Autocomplete, Reorder List, and Slider

VuGen supports the following frameworks at the engine level. This implies that VuGen will create standard Web Click and Script steps, but not AJAX specific functions:

- ▶ Prototype 1.6
- ▶ Google Web Toolkit (GWT) 1.4

Recording an AJAX (Click and Script) Session

To create a Vuser script that emulates AJAX enabled applications, you select the **AJAX (Click and Script)** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions on the Web page, that are affected by AJAX such a reordering a list. For general information about creating and recording a script, see "Recording with VuGen" in *Volume I-Using VuGen*.

You can set event related recording options. See Chapter 17, "Click and Script Recording" in *Volume I-Using VuGen* for more information.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding AJAX (Click and Script) Scripts

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported AJAX controls, VuGen generates a function with an **ajax_xxx** prefix.

In the following example, a user selected item number 1 (index=1) in an Accordion control. VuGen generated an **ajax_accordion** function.

```
web_browser("Accordion.aspx",
            DESCRIPTION,
            ACTION,
            "Navigate=http://labm1app08/AJAX/Accordion/Accordion.aspx",
            LAST);

lr_think_time(5);

ajax_accordion("Accordion",
               DESCRIPTION,
               "Framework=atlas",
               "ID=ctl00_SampleContent_MyAccordion",
               ACTION,
               "UserAction=SelectIndex",
               "Index=1",
               LAST);

web_edit_field("free_text_2",
               "Snapshot=t18.inf",
               DESCRIPTION,
               "Type=text",
               "Name=free_text",
               ACTION,
               "SetValue=FILE_PATH",
               LAST);
```

Note: When you record an AJAX session, VuGen generates standard Web (Click and Script) functions for objects that are not one of the supported AJAX controls. In the example above, the word FILE_PATH was typed into an edit box.

31

AMF Protocol

VuGen allows you to create Vusers that emulate Flash Remoting using the AMF format.

This chapter includes:

- ▶ About Developing AMF Vuser Scripts on page 477
- ▶ Understanding AMF Terms on page 479
- ▶ Working with AMF Functions on page 480
- ▶ Correlating AMF Scripts on page 481
- ▶ Viewing AMF Data on page 485
- ▶ Understanding AMF Scripts on page 485

About Developing AMF Vuser Scripts

Many client applications communicate with servers using RPC (Remote Procedure Calls). RPC, however, presents compatibility and security problems when working over the Internet. Firewalls and proxy servers often block this type of traffic.

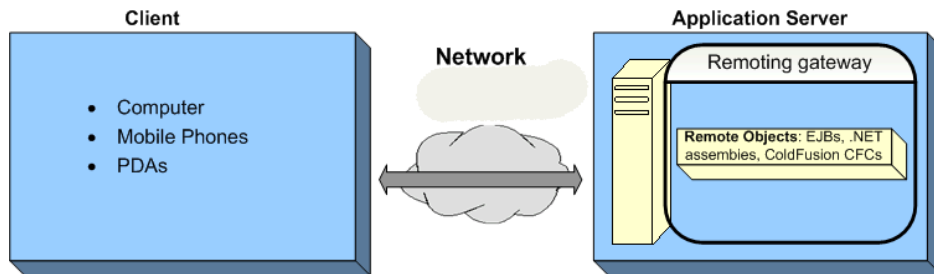
HTTP is supported by all Internet browsers and servers. Therefore, HTTP is a preferred method of communication between client applications and servers when working over the Internet.

SOAP, an XML-based format, provides a secure way to communicate between applications over HTTP. However, since the messages are text-based, SOAP is inefficient when working with large messages such as Flash files and other RIAs (Rich Internet Applications).

To overcome this inefficiency, Macromedia created a proprietary protocol, AMF (Action Messaging Format), which communicates over HTTP in binary format. The binary AMF data set is considerably smaller than that of SOAP's text-based XML.

A typical client application that submits AMF messages to a server is the Flash Player that plays Flash clips on personal computers. The Flash Player sends native Flash objects to an application server via a gateway. The gateway, known as the **Flash Remoting** gateway, is a server-side object, installed on either a Java (including ColdFusion) or .NET server. The gateway acts as a broker that handles requests between the Flash Player and the server. It translates Flash objects into native objects for the server and passes them on to the appropriate server-side services.

After the results are returned, the gateway serializes them back into native Flash objects and sends them to the Flash client via AMF.



To create a Vuser script that emulates the AMF traffic, you select the **AMF** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions against the Web server. For general information about creating and recording a script, see "Recording with VuGen" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding AMF Terms

The following table provides definitions for the most common terms that relate to AMF:

Term/ Abbreviation	Description
ActionScript	A script programming language used for controlling Flash movies and applications. Its syntax is similar to JavaScript.
AMF	A proprietary binary communication protocol used for Flash Remoting.
Flash Remoting	Flash Remoting allows data to be exchanged between a Flash Player and an application server using the AMF format.
Flex	An application server for generating RIAs (Reach Internet Applications).
SOAP	A standard for exchanging XML-based messages over a computer network, normally using HTTP.

Working with AMF Functions

When you record a Flash Remoting session, VuGen generates AMF Vuser script functions that emulate your actions. All AMF functions begins with an **amf** prefix.

In the following example, the **amf_define_header_set** function defines a header set. The **amf_call** function accesses a gateway and sends a message to the server.

```
amf_define_header_set("Id=amf_header_set",
    HEADER,
    "Name=amf_server_debug",
    "MustUnderstand=true",
    "Data=<object><boolean key=\"coldfusion\">true</boolean>
        <boolean key=\"\"amfheaders\">false</boolean>...
    LAST);

amf_call("flashgateway.samples.FlashJavaBean.testDocument",
    "Gateway=http://testlab:8200/flashservices/gateway",
    "AMFHeaderSetId=amf_header_set",
    "Snapshot=t3.inf",
    MESSAGE,
    "Method=flashgateway.samples.FlashJavaBean.testDocument",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    "<xmlString><![CDATA[<TEST message=\"test\"><INSIDETEST
        /></TEST>]]></"
    "xmlString>",
    END_ARGUMENTS,
    LAST);
```

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

Correlating AMF Scripts

Flash Applications often contain dynamic data, data that changes each time you run the script. For example, certain servers use links comprised of the current date and time. Alternatively, the object name may change from run to run.

When you record a Vuser script, VuGen records a set of data and argument values. When you replay the script, however, the server may reject these arguments and issue an error. This error could be the result of dynamic data that is outdated and no longer accepted by the server.

To overcome this, you apply correlation to your script:

- ▶ Save the server response in preparation for extracting the required values.
- ▶ Extract the required values from the server response.
- ▶ Save the values to a parameter.
- ▶ Use those parameters as input to your AMF requests.

These errors are not always obvious, and you may only detect them by carefully examining Vuser log files. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

To perform correlation:

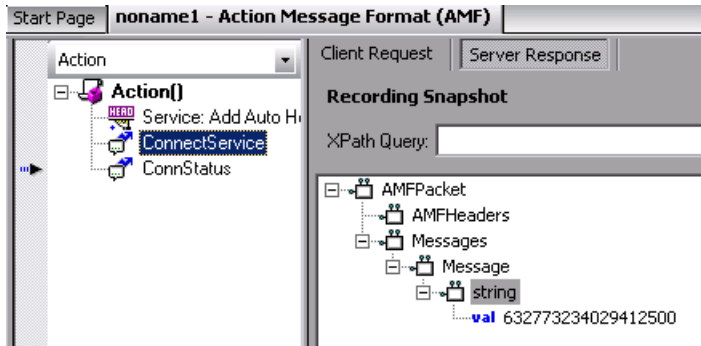
- 1** Locate the step in your script that failed due to dynamic values that need correlation.

Use the Replay Log to assist you in finding the problematic step.

```
-----
Action.c(16): Error: Server returned error for message #1 : "Incorrect session ID sent"
Action.c(16): There was an error during the AMF Call ("ConnStatus")
```

2 Identify the server response with the correct value in one of the previous steps.

Examine the proceeding steps in Tree View and look for the value in the **Server Response** tab.



3 Save the entire server response to a parameter.

Before you extract the value, the entire server response should be saved to a parameter as follows:

- Right-click the step node (in the Action pane) corresponding to the server response containing the value and select **Properties**.
- In the AMF Call Properties dialog, type a **Response parameter** name. For details, see "AMF Call Properties" on page 486.
- Click **OK** to save the new parameter name.

4 Save the original server response value to a parameter.

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

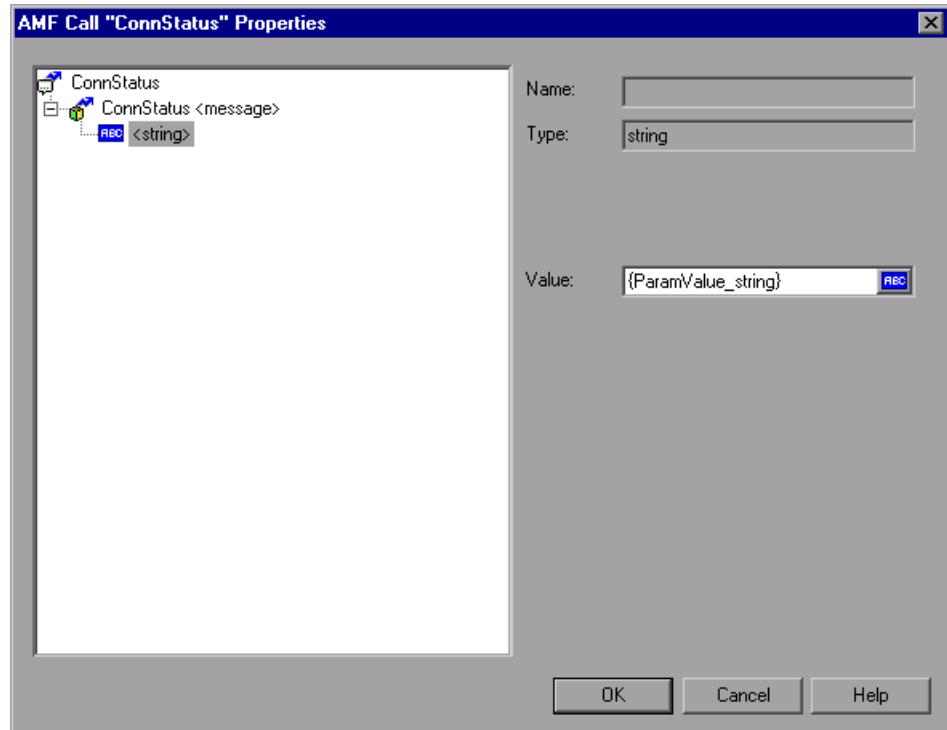


- In the XML Parameter Properties dialog, specify a parameter **Name**. You will use this name in subsequent steps.
- Click **OK**. The script will now contain a new function, **lr_xml_get_values**.

5 Insert the parameter in the subsequent calls.

In VuGen edit view, beginning with the call that failed, replace the value in all subsequent calls to the object with the parameter that you defined:

- Right-click the step node (in the Action pane) corresponding to the failed call and select **Properties**.
- Locate the argument that required correlation.
- In the **Value** box, type the parameter name in curly brackets, for example, **{ParamValue_string}**.



- Click **OK**.

6 Run the script.

Make sure that VuGen properly substitutes the argument value with the parameter value that you saved.

In the following example, `lr_xml_get_values` retrieves the value from the response and creates the parameter `ParamValue_string`. This parameter, `ParamValue_string`, was used in the next `amf_call` function.

```

amf_call(
    "ConnectService",
    "Gateway=http://labm1app08/AMF/EchoAMF/gateway.aspx",
    "Snapshot=t6.inf",
    "ResponseParameter=resp",
    MESSAGE,
    "Method=EchoAMF.SpecialCases.ConnectService",
    "TargetObjectId=/1",
    BEGIN_ARGUMENTS,
    END_ARGUMENTS,

    LAST);

lr_xml_get_values("XML={resp}",
    "FastQuery=/AMFPacket/Messages/Message/string",
    "ValueParam=ParamValue_string",
    LAST);

amf_call(
    "ConnStatus",
    "Gateway=http://labm1app08/AMF/EchoAMF/gateway.aspx",
    "Snapshot=t7.inf",
    MESSAGE,
    "Method=EchoAMF.SpecialCases.ConnStatus",
    "TargetObjectId=/2",
    BEGIN_ARGUMENTS,
    "<string>{ParamValue_string}</string>",
    END_ARGUMENTS,

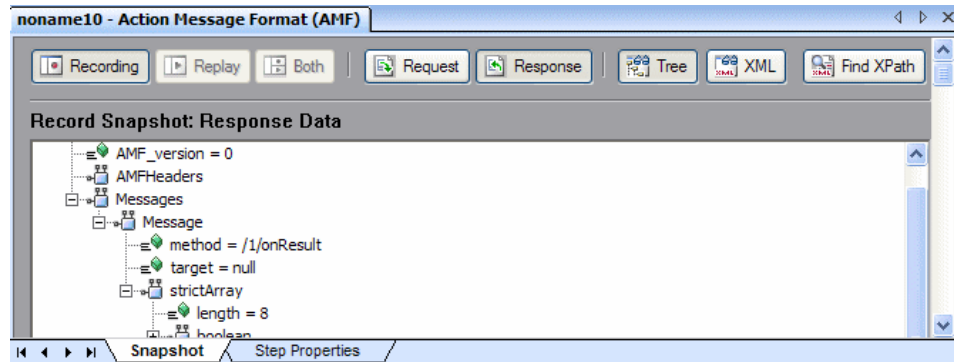
    LAST);

```

For more information about correlation for Web Users, see Chapter 42, "Web (HTTP/HTML) Correlation Rules."

Viewing AMF Data

To view the AMF data in XML format, select the appropriate step in the Tree View. The right pane of VuGen displays the client request and server response in XML hierarchy. To view it in true XML, click the **XML** button.

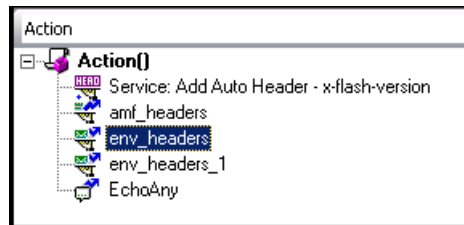


To view all the elements of the message, make sure that the nodes of the XML are expanded. To view the XML argument values in a grid, select **Show node values in grid**.

VuGen provides a utility to find XML through its XPath. You can also use the Query Builder to assist you in creating the queries. For more information, see "Querying an XML Tree" on page 499.

Understanding AMF Scripts

In the following example, the AMF script contains an AMF call, AMF header, and AMF envelope header. To view details of each node, you select a node and select Properties from the right-click menu.

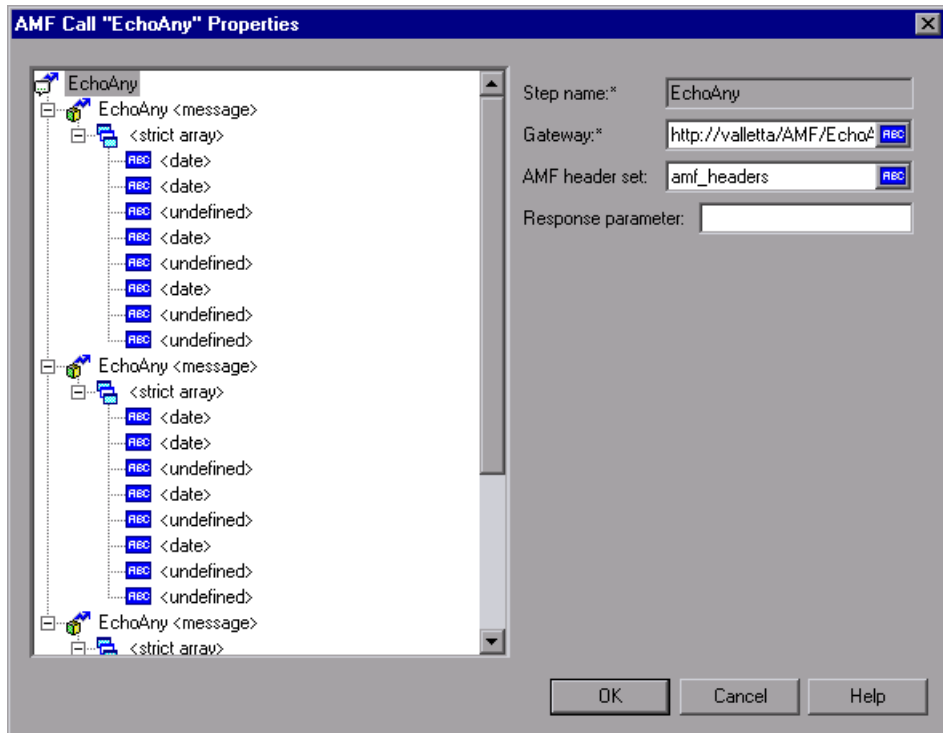


The following sections describe the AMF node properties:

- ▶ AMF Call Properties
- ▶ AMF Header Set Properties
- ▶ AMF Envelope Header Set Properties

AMF Call Properties

The AMF Call Properties dialog shows the AMF request details: the AMF call, one or more messages, and the arguments for each message. In the example below, the AMF call is **EchoAny**, and it is associated with the **amf_headers** header set.

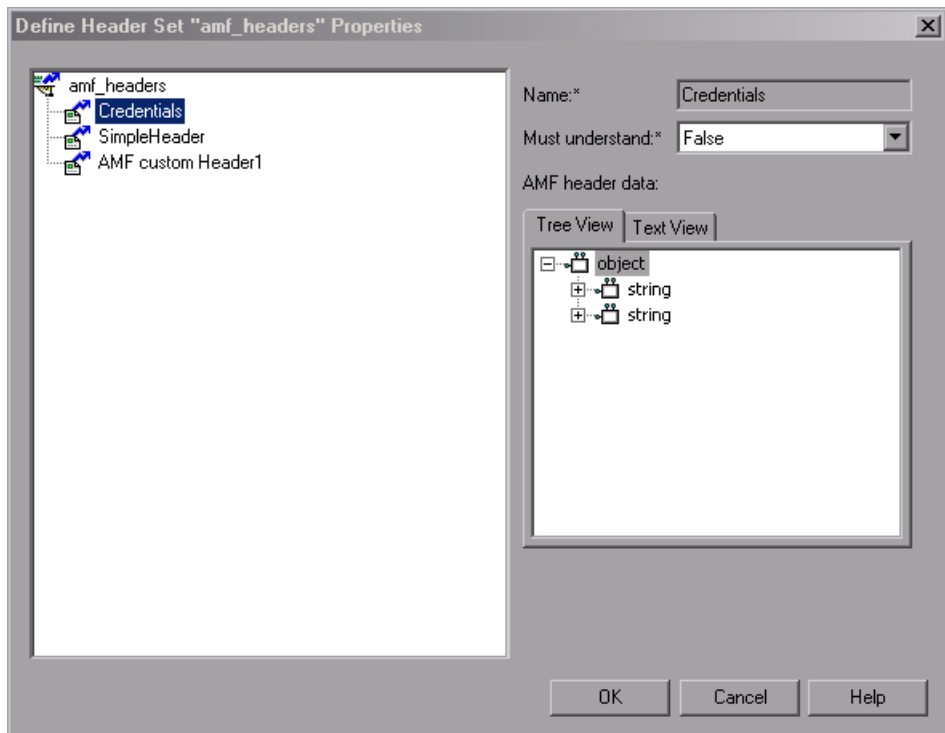


Each AMF call has three levels:

- ▶ The AMF Call level contains a step name, gateway, AMF header set, and response parameter
- ▶ The AMF Message level contains a method, target object ID, and Envelope header set
- ▶ The AMF Argument level contains a name, type, and value

AMF Header Set Properties

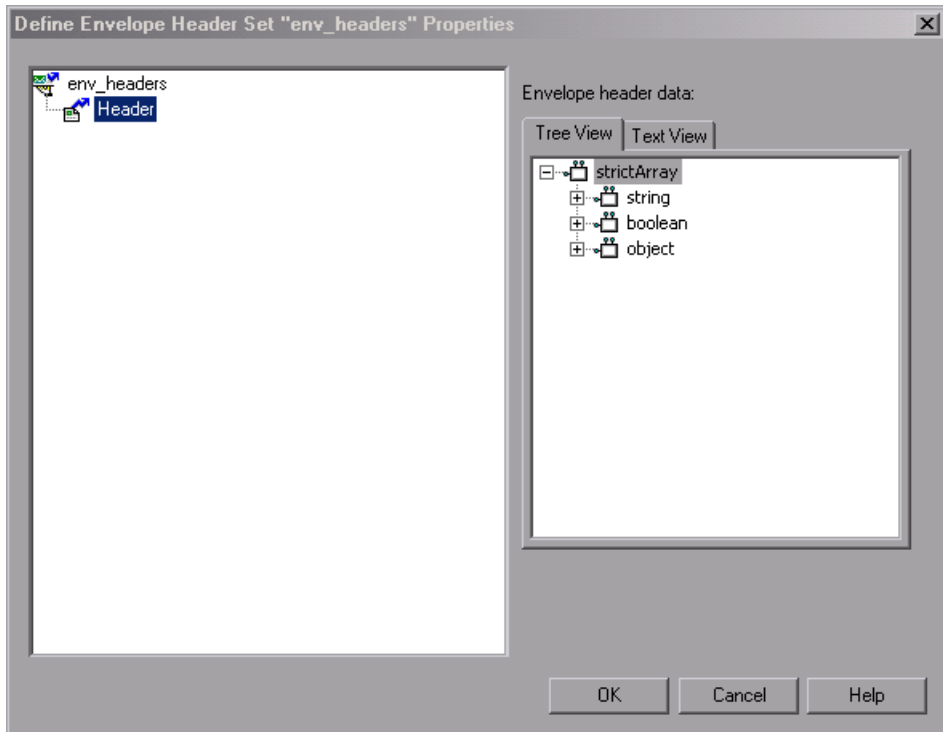
The AMF Header Set Properties dialog shows the AMF header set definition: One or more AMF header names and associated header data in XML format. In the example below, the header set has three headers.



Each header has a **Must understand** argument which indicates whether processing the call continues if the server cannot interpret the header. If MustUnderstand is true, this header is mandatory and processing is aborted if it is not understood.

AMF Envelope Header Set Properties

The AMF Envelope Header Set Properties dialog shows the AMF envelope header set and corresponding data for each header.



32

FTP Protocol

VuGen allows you to emulate network activity by directly accessing an FTP (File Transfer Protocol) server.

This chapter includes:

- ▶ About Developing FTP Vuser Scripts on page 489
- ▶ Working with FTP Functions on page 490

About Developing FTP Vuser Scripts

The FTP protocol is a low-level protocol that allows you to emulate the actions of a user working against an FTP server.

For FTP, you emulate users logging into an FTP server, transferring files, and logging out. To create a script, you can record an FTP session or manually enter FTP functions.

When you record an FTP session, VuGen generates functions that emulate the mail client's actions. If the communication is performed through multiple protocols such as FTP, HTTP, and a mail protocol, you can record all of them. For instructions on specifying multiple protocols, see "Recording with VuGen" in *Volume I-Using VuGen*.

To create a script for the FTP protocol, you select the FTP protocol type in the E-Business category. To begin recording, you click the **Record** button and perform typical actions against the FTP server. For more information on creating and recording a script, see "Recording with VuGen" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availability Center* documentation.

Working with FTP Functions

When recording a FTP session, VuGen generates FTP functions. Each FTP function begins with an **ftp** prefix.

Most FTP functions come in pairs—one for global sessions and one where you can indicate a specific mail session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the **ex** suffix. For example, **ftp_logon** logs on to the FTP server globally, while **ftp_logon_ex** logs on to the FTP server for a specific session.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

You can indicate the programming language in which to create a Vuser script. For more information, see "Setting Script Generation Preferences" in *Volume I-Using VuGen*.

In the following example, the `ftp_delete` function deletes the `test.txt` file from the FTP server.

```
ftp_logon("FTP",
          "URL=ftp://user:pwd@ftp.merc-int.com",
          "LocalAddr=ca_server:21",
          LAST);

ftp_delete("Ftp_Delete",
          "PATH=/pub/for_jon/test.txt", ENDITEM,
          LAST);

ftp_logout();
```

33

Flex Protocol

VuGen allows you to create Vusers that emulate the protocol suite provided with the Flex 2 SDK.

This chapter includes:

- ▶ About Developing Flex Vuser Scripts on page 491
- ▶ Working with Flex Functions on page 493
- ▶ Correlating Flex Scripts on page 494
- ▶ Viewing Flex Data on page 498
- ▶ Setting Flex Step Properties on page 501
- ▶ Working with Flex RTMP on page 502

About Developing Flex Vuser Scripts

Flex is a collection of technologies that provide developers with a framework for building RIAs (Rich Internet Applications) based on the Flash Player.

RIAs are lightweight online programs that provide users with more dynamic control than with a standard web page. Like Web applications built with AJAX, Flex applications are more responsive, because the application does not need to load a new Web page every time the user takes action. However, unlike working with AJAX, Flex is independent of browser implementations such as JavaScript or CSS. The framework runs on Adobe's cross-platform Flash Player.

Flex 2 applications consist of many MXML and ActionScript files. They are compiled into a single SWF movie file which can be played by Flash player, installed on the client's browser.

Flex 2 supports a variety of client/server communication methods, such as RPC, Data Management, and Real-Time messaging. It supports several data formats such as HTTP, AMF, and SOAP.

VuGen lets you create a Vuser script that emulates communication with Flex 2 RPC services. VuGen's **Flex** type lets you create scripts that emulate Flex applications working with AMF3 or HTTP data. For Flex applications working with SOAP data, use the **Web Services** Vuser type.

The following sections describe how to create scripts for applications using AMF3 or HTTP communication. For information about Web Service Vuser scripts, see Chapter 2, "Web Services Protocol."

To create a script, you select the **Flex** protocol type from the **E-Business** category. To begin recording, click the **Record** button and perform typical actions in your Flex application. For general information about creating and recording a script, see "Recording with VuGen" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Working with Flex Functions

When you record a Flex application, VuGen generates Flex Vuser script functions that emulate your application. The following functions represent some of the Flex remoting steps:

Function Name	Description
flex_login	Logs on to a password-protected Flex application.
flex_logout	Logs off of a password-protected Flex application.
flex_ping	Checks if a Flex application is available.
flex_remoting_call	Invokes one or more methods of a server-side Remote object (RPC).
flex_web_request	Sends an HTTP request with any method supported by HTTP.
flex_amf_call	Sends an AMF request.
flex_amf_define_header_set	Defines a set of AMF headers.
flex_amf_define_envelope_header_set	Defines a set of envelope headers.

In the following example, **flex_ping** checks for the availability of a service. The **flex_remoting_call** function invokes the service remotely.

```

flex_ping("1",
  "URL=http://testlab1/weborb30/console/weborb.aspx",
  "Snapshot=t6.inf",
  LAST);

flex_remoting_call("getProductEdition::GenericDestination",
  "URL=http://testlab1/weborb30/console/weborb.aspx",
  "Snapshot=t7.inf",
  INVOCATION,
  "Target=/2",
  "Operation=getProductEdition",
  "Destination=GenericDestination",
  "DSEndpoint=my-amf",
  "Source=Weborb.Management.LicenseService",
  "Argument=<arguments/>",
  LAST);

```

For detailed syntax information about all of the Flex functions, see the *Online Function Reference* (**Help > Function Reference**).

Correlating Flex Scripts

Flex applications often contain dynamic data, data that changes each time you run the script. For example, the object name may change from run to run.

When you record a Vuser script, VuGen records a set of data and argument values. When you replay the script, however, the server may reject these arguments and issue an error. This error could be the result of dynamic data that is outdated and no longer accepted by the server.

To overcome this, you apply correlation to your script:

- ▶ Save the server response in preparation for extracting the required values.
- ▶ Extract the required values from the server response.
- ▶ Save the values to a parameter.
- ▶ Use those parameters as input to your Flex requests.

These errors are not always obvious, and you may only detect them by carefully examining Vuser log files. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

To perform correlation:

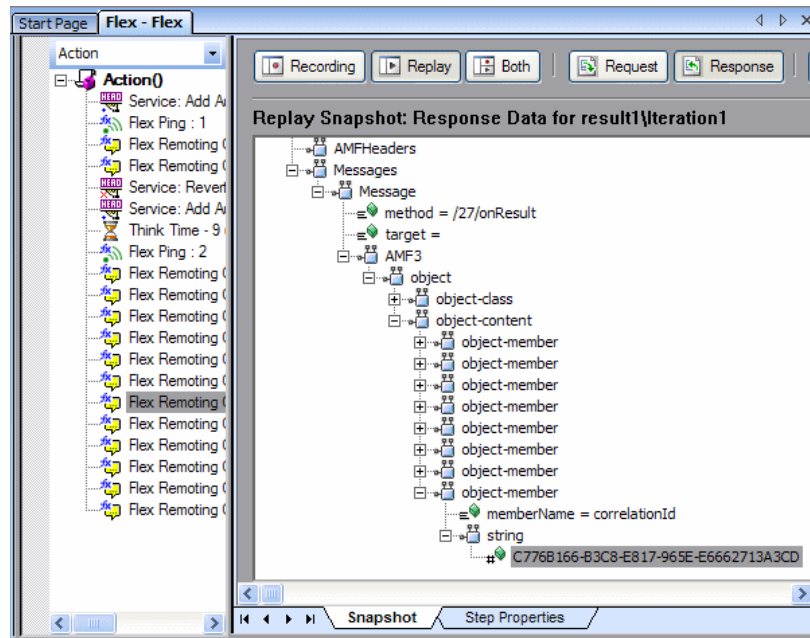
1 Locate the step in your script that failed due to dynamic values that need correlation.

Use the Replay Log to assist you in finding the problematic step.

Action.c(16): Error Server returned error for message #1 : "Incorrect session ID sent"/
Action.c(16): There was an error during the Flex Call ("ConnStatus")

2 Identify the server response with the correct value in one of the previous steps.

Double-Click the error in the Replay log to go to the step with the error. Examine the preceding steps in Tree View and look for the value in the **Server Response** tab.



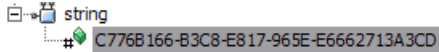
3 Save the entire server response to a parameter.

Before you extract the value, the entire server response should be saved to a parameter as follows:

- Right-click the step node (in the left Action pane) corresponding to the server response containing the value and select **Properties**.
- In the Flex Call Properties dialog box, type a **Response parameter** name.
- Click **OK** to save the new parameter name.

4 Save the original server response value to a parameter.

- In the **Replay Snapshot: Response Data**, right-click the node above the value (for example, string), and select **Save value in parameter**.



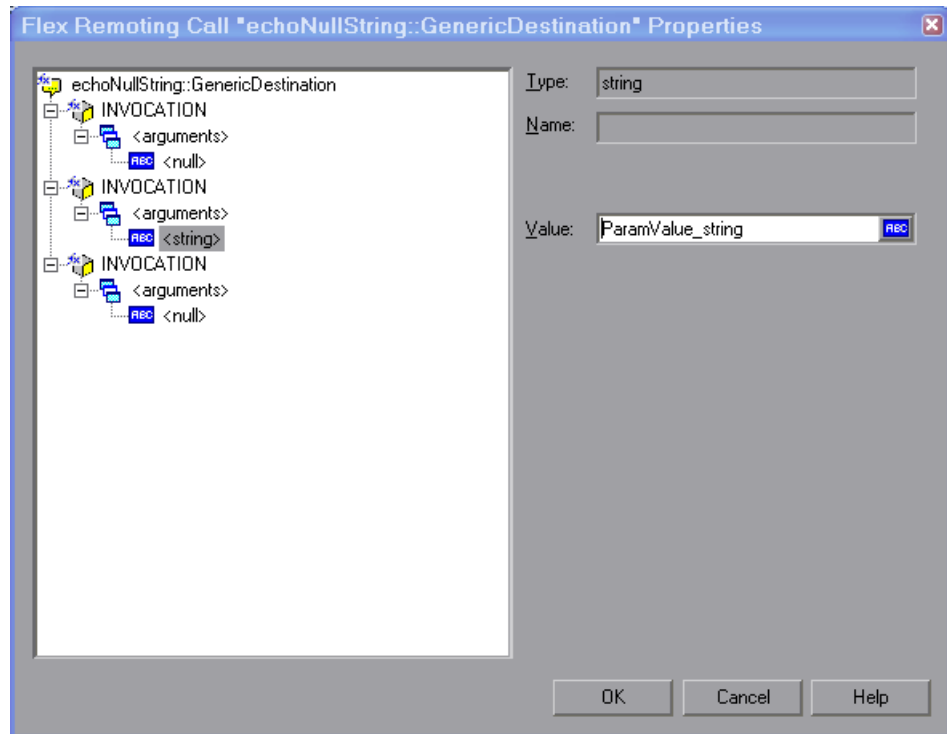
- In the XML Parameter Properties dialog, specify a parameter **Name**. You will use this name in subsequent steps.
- Click **OK**. The script will now contain a new function, **lr_xml_get_values**.

5 Insert the parameter in the subsequent calls.

In VuGen edit view, beginning with the call that failed, replace the value in all subsequent calls to the object with the parameter that you defined:

- Right-click the step node (in the Action pane) corresponding to the failed call and select **Properties**.
- Locate the argument that required correlation.

- In the **Value** box, type the parameter name in curly brackets, for example, **{ParamValue_string}**.



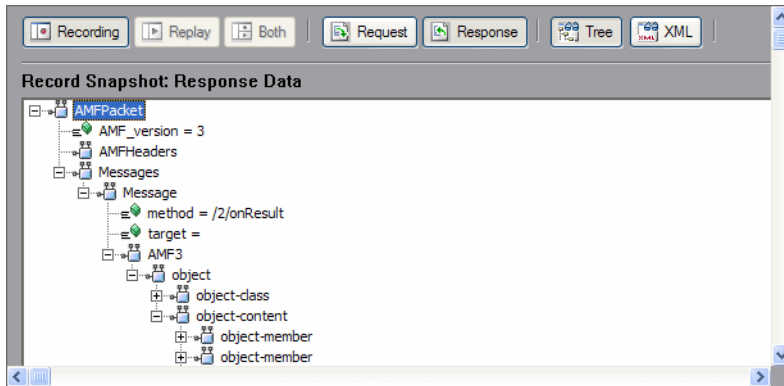
Click **OK**.

6 Run the script.

Make sure that VuGen properly substitutes the argument value with the parameter value that you saved.

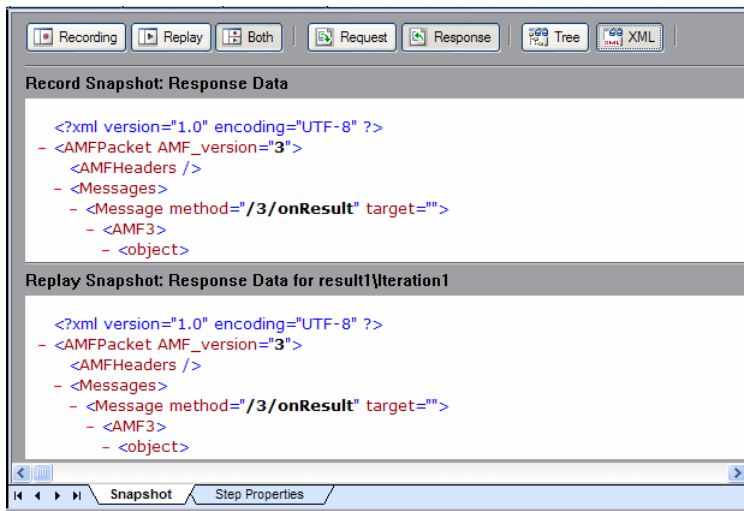
Viewing Flex Data

To view the Flex data, select the appropriate step in the Tree View. The right pane of VuGen displays a snapshot of the data.



To view all the elements of the message, expand the desired nodes. You can view the Request or Response data for Recording and Replay, by clicking the appropriate buttons in the snapshot.

To view the data in its XML structure, click the **XML** button in the snapshot.



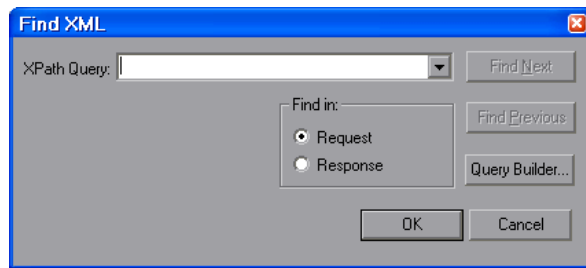
Querying an XML Tree

VuGen provides a Query Builder that lets you create and execute queries on the XML.

VuGen displays the XML code in an expandable tree. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

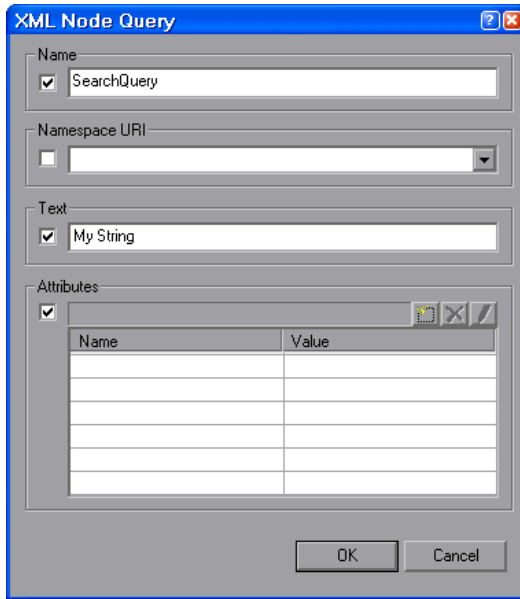
To perform a query:

- 1 In the Snapshot tab, select on the node that you want to search. Click the **Find XML** button. The Find XML dialog button opens.



- 2 Select Request or Response. Enter an XPath query and click **OK**. To formulate a query, click **Query Builder** button. The XML Node Query dialog box opens.

3 Enable one or more items for searching.



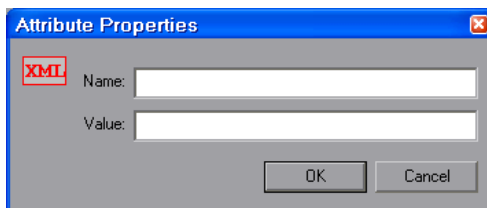
4 Enable the **Name** section to search for the name of a node or element.

5 Enable the **Namespace URI** section to search for a namespace.

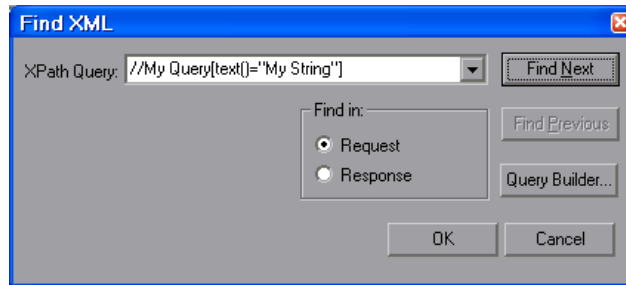
6 Enable the **Text** section to search for the value of the element indicated in the Name box.

7 Enable the **Attributes** section to search for an attribute.

8 Enter the search text in the appropriate boxes. To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.



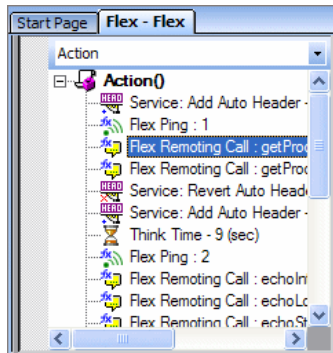
- 9 Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the Find XML box.



- 10 Click **Find Next** to begin the search.

Setting Flex Step Properties

The Flex Vuser script contains a series of Flex calls. You can view information about each step by selecting it in the Tree view.



The right pane has two tabs (for most Flex steps)—Snapshot and Properties. The Snapshot tab shows the data, while the Properties step displays the properties each step:

The following example shows the properties for a **Flex Remoting Call** step.



As you select each node, the right pane shows the properties specific for that node.

Working with Flex RTMP

Flex can record and replay scripts involving RTMP (Real Time Messaging Protocol). In order to enable RTMP simulation, you must configure the recording options for the Flex protocol.

To enable RTMP:

- 1** Open the Recording Options dialog box by selecting **Tools > Recording Options** or clicking the **Options** button in the Start Recording dialog box.
- 2** In the **Network > Port Mapping** node click **Options**.
- 3** Set the **Send-Receive buffer size threshold** to 1500.

34

LDAP Protocol

VuGen allows you to emulate the communication with an LDAP server.

This chapter includes:

- About Developing LDAP Vuser Scripts on page 503
- Working with LDAP Functions on page 504
- Defining Distinguished Name Entries on page 506
- Specifying Connection Options on page 507

About Developing LDAP Vuser Scripts

LDAP, the Lightweight Directory Access Protocol, is a protocol used to access a directory listing. The LDAP directory is composed of many LDAP entries. Each LDAP entry is a collection of attributes with a name, called a distinguished name (DN). For more information about DN, see "Defining Distinguished Name Entries" on page 506.

LDAP directory entries are arranged in a hierarchical structure that reflects political, geographic, and/or organizational boundaries. Entries representing countries appear at the top of the tree. Below them are entries representing states or national organizations. Below them might be entries representing people, organizational units, printers, documents, or just about anything else.

VuGen records communication over LDAP servers. It creates a script, with functions that emulate your actions. This includes logging in and out of the server, adding and deleting entries, and querying an entry.

To create a script for the LDAP protocol, you select the LDAP protocol type in the E-Business category. To begin recording, select **Vuser > Start Recording**, and perform typical actions against the LDAP server. For more information on the recording procedure, see "Recording with VuGen" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availability Center* documentation.

Working with LDAP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see "Setting Script Generation Preferences" in *Volume I-Using VuGen*.

LDAP Vuser script functions emulate the LDAP protocol. Each LDAP function begins with the **ldap** prefix.

All LDAP functions come in pairs—one for global sessions and one where you can indicate a specific session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the **ex** suffix. For example, **ldap_logon** logs on to the LDAP server globally, while **ldap_logon_ex** logs on to the LDAP server for a specific session.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the user logs on to an LDAP server, ldap1. It adds an entry and then renames the OU attribute from Sales to Marketing.

```
Action()
{
    // Logon to the LDAP server
    mldap_logon("Login",
               "URL=ldap://johnsmith:tiger@ldap1:80",
               LAST);

    // Add an entry for Sally R. Jones
    mldap_add("LDAP Add",
             "DN=cn=Sally R. Jones,OU=Sales, DC=com",
             "Name=givenName", "Value=Sally", ENDITEM,
             "Name=initials", "Value=R", ENDITEM,
             "Name=sn", "Value=Jones", ENDITEM,
             "Name=objectClass", "Value=contact", ENDITEM,
             LAST);

    // Rename Sally's OU to Marketing
    mldap_rename("LDAP Rename",
                "DN=CN=Sally R. Jones,OU=Sales,DC=com",
                "NewDN=OU=Marketing",
                LAST);

    // Logout from the LDAP server
    mldap_logoff();

    return 0;
}
```

Defining Distinguished Name Entries

The LDAP API references objects by its **distinguished name** (DN). A DN is a sequence of **relative distinguished names** (RDN) separated by commas.

An RDN is an attribute with an associated value in the form attribute=value. The attribute names are not case-sensitive. The following table lists the most common RDN attribute types.

String	Attribute Type
DC	domainComponent
CN	commonName
OU	organizationalUnitName
O	organizationName
STREET	streetAddress
L	localityName
ST	stateOrProvinceName
C	countryName
UID	userid

The following are examples of distinguished names:

DN=CN=John Smith,OU=Accounting,DC=Fabrikam,DC=COM

DN=CN=Tracy White,CN=admin,DC=corp,DC=Fabrikam,DC=COM

The following table lists reserved characters that cannot be used in an attribute value.

Character	Description
	space or # character at the beginning of a string
	space character at the end of a string
,	comma

+	plus sign
"	double quote
\	backslash
<	left angle bracket
>	right angle bracket
;	semicolon

To use a reserved character as part of an attribute value, you must precede it with an escape character, a backslash (\). If an attribute value contains other reserved characters, such as the equal sign (=) or non-UTF-8 characters, you must encode it in hexadecimal format—a backslash followed by two hex digits.

The following are examples of DNs that include escaped characters. The first example is an organizational unit name with an embedded comma; the second example is a value containing a carriage return.

```
DN=CN=Bitwise,OU=Docs\, Support,DC=Fabrikam,DC=COM
DN=CN=Before\0DAfter,OU=Test,DC=North America,DC=Fabrikam,DC=COM
```

Specifying Connection Options

Using the `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**).

35

Microsoft .NET Protocol

VuGen records applications that were created in the .NET Framework environment.

This chapter includes:

- ▶ About Recording Microsoft .NET Vuser Scripts on page 510
- ▶ Getting Started with Microsoft .NET Vusers on page 511
- ▶ Viewing Scripts in VuGen and Visual Studio on page 513
- ▶ Viewing Data Sets and Grids on page 515
- ▶ Correlating Microsoft .NET Scripts on page 516
- ▶ Configuring Application Security and Permissions on page 519
- ▶ Recording WCF Duplex Communication on page 523

Before recording a script, we recommend that you read Chapter 36, "Microsoft .NET Filters." These guidelines will help you create an optimal script that accurately emulates your application.

About Recording Microsoft .NET Vuser Scripts

Microsoft .NET Framework provides a solid foundation for developers to build various types of applications such as ASP.NET, Windows Forms, Web Services, distributed applications, or applications that combine several of these models.

VuGen supports .NET as an application level protocol. It allows you to create Vuser scripts that emulate users of Microsoft .NET client applications created in its .NET Framework. VuGen records all of the client actions through methods and classes, and creates a script in C Sharp or VB .NET.

By default, the VuGen environment is configured for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation) applications. Contact Customer Support for information on how to configure VuGen to record applications created with other client-server activity.

For more information about .NET and the above environments, see the MSDN Web site, <http://msdn2.microsoft.com>.

Limitations

The following limitations apply to the VuGen recording of a Microsoft .NET application:

- ▶ Microsoft .NET scripts only support single-protocol recording in VuGen.
- ▶ Direct access to public fields is not supported—the AUT must access fields through methods or properties.
- ▶ VuGen does not record static fields in the applications—it only records methods within classes.
- ▶ Multi-threaded support is dependent on the client application. If the recorded application supports multi-threading, then the Vuser script will also support multi-threading.
- ▶ In certain cases, you may be unable to run multiple iterations without modifying the script. Objects that are already initialized from a previous iteration, cannot be reinitialized. Therefore, to run multiple iterations, make sure to close all of the open connections or remoting channels at the end of each iteration.

- ▶ Recording is not supported for Enterprise Services communication based on MSMQ and Enterprise Services hosted in IIS.
- ▶ VuGen partially supports the recording of WCF services hosted by the client application.
- ▶ Recording is not supported for Remoting calls using a custom proxy.
- ▶ Recording is not supported for **ExtendedProperties** property of ADO.NET objects, when using the default ADO.NET filter.
- ▶ Applications created with .NET Framework 1.1 which are not compatible with Framework 2.0, cannot be recorded. To check if your Framework 1.1 application is compatible, add the following XML tags to your application's .config file:

```
<configuration>
  <startup>
    <supportedRuntime version="v2.0.50727"/>
  </startup>
</configuration>
```

Invoke the application (without VuGen) and test its functionality. If the application works properly, VuGen can record it. Remove the above tags before recording the AUT with VuGen. For more information regarding this solution, see the MSDN Knowledge Base.

Getting Started with Microsoft .NET Vusers

This section describes the process of developing Microsoft .NET Vuser scripts. Note that the AUT (Application Under Test) must be installed on the machine running the script and all load generators.

To develop a basic .NET Vuser script:

1 Record the application using VuGen.

Start VuGen and create a new Vuser script. Specify **Microsoft .NET** as the type of Vuser. Select an application to record and set the recording options. For more information, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

For general details about recording, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Debug the script and set the filter.

Examine the recorded script and determine if the required methods were recorded. If necessary, modify the filter and record the script again.

For details about setting the filter, see "Guidelines for Setting Filters" on page 535.

3 Correlate the script.

Correlate the values in your script to save them as parameters for use at a later point in the script.

For details, see "Correlating Microsoft .NET Scripts" on page 516.

4 Run the script from VuGen.

Save and run the script in VuGen to verify that it runs correctly. To insure a successful replay, compile and run the script in VuGen before running it remotely on a Load Generator machine or via the Controller. This applies each time you make a change to the script—either in the script's settings or the actual script. In addition, if you save the script under another name, replay it in VuGen before running it on a Load Generator.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

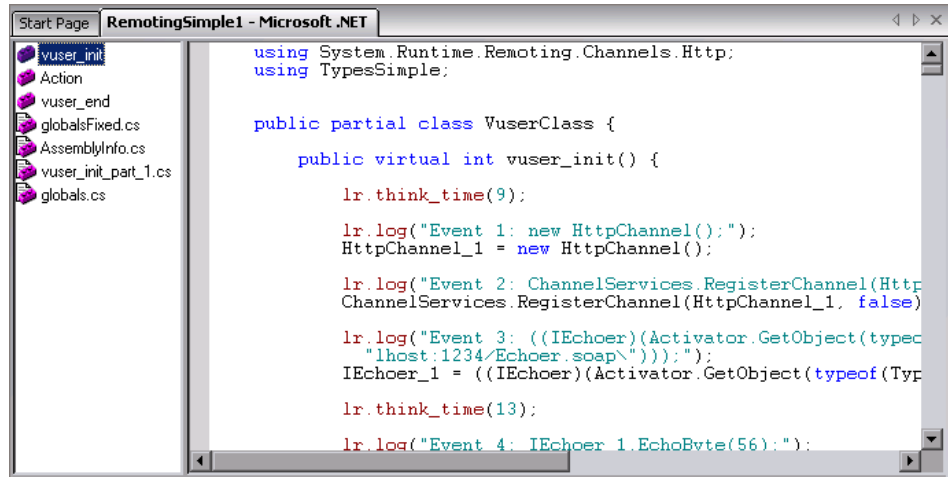
The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Viewing Scripts in VuGen and Visual Studio

After the recording, you can view the script in VuGen's Script view.



The screenshot shows the VuGen Script view for a project named 'RemotingSimple1 - Microsoft .NET'. The left sidebar displays a tree view with files: vuser_init, Action, vuser_end, globalsFixed.cs, AssemblyInfo.cs, vuser_init_part_1.cs, and globals.cs. The main editor area displays the following C# code:

```
using System.Runtime.Remoting.Channels.Http;
using TypesSimple;

public partial class VuserClass {
    public virtual int vuser_init() {
        lr.think_time(9);

        lr.log("Event 1: new HttpChannel()");
        HttpChannel_1 = new HttpChannel();

        lr.log("Event 2: ChannelServices.RegisterChannel(Http
ChannelServices.RegisterChannel(HttpChannel_1, false)

        lr.log("Event 3: ((IEchoer)(Activator.GetObject(typec
"localhost:1234/Echoer.soap\")));");
        IEchoer_1 = ((IEchoer)(Activator.GetObject(typeof(Typ

        lr.think_time(13);

        lr.log("Event 4: IEchoer 1.EchoByte(56):");
```

VuGen records the actions that occurred and generates C Sharp or VB code. By default, VuGen wraps all of the steps in **lr.log** calls.

When you replay the script, VuGen compiles it first to make sure that all of the calls are valid and that the syntax is correct. VuGen compiles the script into a DLL file, **Script.dll**, saved in the script's **bin** folder. This DLL file contains three functions - Init, Actions, and End.

You can compile the script to check its syntax, without running it. To compile the script directly from VuGen, click Shift+F5 or select **Vuser > Compile**. If VuGen detects a compilation error, it displays it in the Output window. Double-click on the error to go to the problematic line in the script.

To run the script directly from VuGen, click F5 or select **Vuser > Run**. Breakpoints and step-by-step replay are not supported in VuGen's editor window for Microsoft .NET Vusers. To debug a script and run it with breakpoints or step-by-step, run it from within Visual Studio .NET as described below.

Viewing Scripts in Visual Studio

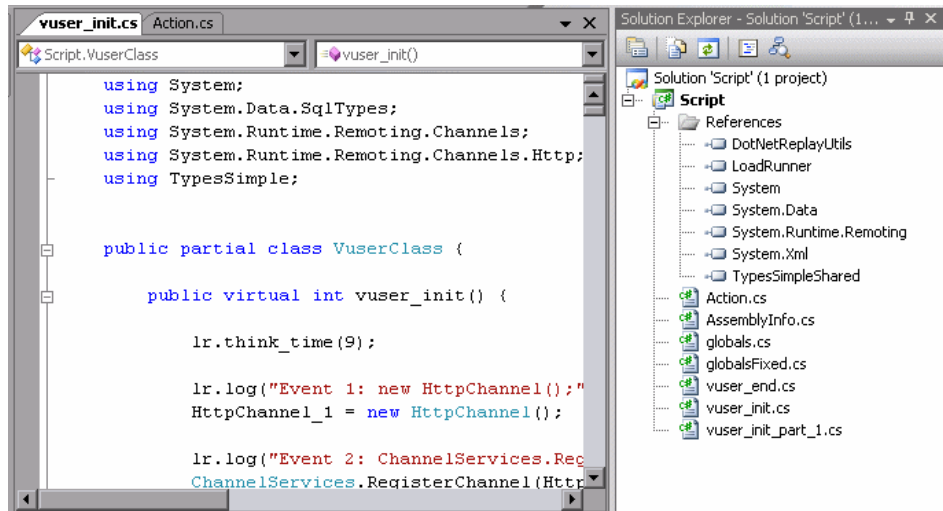
Visual Studio provides you with additional tools to view, edit, and debug your script. You can add breakpoints, view variable values, add assembly references, and edit the script using Visual Studio's IntelliSense. You can also run the script in a step-by-step mode for debugging.

When you save your script, VuGen creates a Visual Studio 2005 solution file, **Script.sln**, in your script's folder. You can open the solution file in Visual Studio .NET and view all of its components in the Solution Explorer.



To open the solution in Visual Studio 2005, select **Vuser > Open Solution in Visual Studio** or click the Visual Studio button on VuGen's toolbar.

Double-click the appropriate section in the Solution Explorer, such as **vuser_init.cs**, to view the contents of the script.



Note that VuGen automatically loads all of the necessary references that were required during recording. You can add additional references for use during compilation and replay through the Solution Explorer. Select the **Reference** node and select **Add Reference** from the right-click menu.

Click on **globals.cs** or **globals.vb** in the Solution Explorer to view a list of the variables defined and used by your script.

Viewing Data Sets and Grids

When you record a method returning a dataset, data table, or data reader action, VuGen generates a grid for displaying the data.

When working with a data reader, VuGen collects the data retrieved from each **Read** operation and converts it to the replay helper function, **DoDataRead**.

For example, after recording the following application code,

```
SqlDataReader reader = command.ExecuteReader();
while( reader.Read() )
{
    // read the values, e.g., get the string located in column 1
    string str = reader.GetString(1)
}
```

VuGen generates the following lines in the script:

```
SqlDataReader_1 = SqlCommand_1.ExecuteReader();
LrReplayUtils.DoDataRead(SqlDataReader_1, out valueTable_1, true, 27);
```

where two the parameters indicate that during recording, the Application read all 27 available records. Therefore, during replay the script will read all available records.

In addition, VuGen generates a data grid containing all the information retrieved by the **Read** operations.

During replay you can use the output data table, containing the actual retrieved values, for correlation and verification. For more information regarding the **DoDataRead** function, see the *Online Function Reference* (**Help > Function Reference**).

By default, VuGen displays the grids in your script. To disable the grid display and instruct VuGen to show the collapsed version of the grid, select **View > Data Grids**.

	FLIGHT NUMBER	DEPARTURE INITIALS	DEPARTURE	DAY OF WEEK	ARRIVAL INITIALS	ARRIVAL	DEPARTURE TIME
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.

For additional information about grids, see "Working with Grids" on page 366.

Correlating Microsoft .NET Scripts

After you record a session, you may need to **correlate** one or more values within your script. Correlating a value means that you capture a value during the script replay, and save it to a parameter. You can then use this parameter at a later point in the script.

VuGen automatically does a basic correlation—if an object is returned from a function call and later called in the script or if the object is passed to another method, VuGen uses the same object instance.

You can further correlate values in your script by manually saving the parameters through coding, or through VuGen's built-in correlation tools.

To correlate a value within VuGen, you locate the value in your dataset and use the right-click menu to save the value to a parameter.

To correlate a value for ADO.NET environments:

1 Locate the dataset in your script.

Display the grids in your script to show the returned datasets. If the grids are not visible, select **View > Data Grids** or expand the applicable **DATASET_XML** statement. For example:

	CustomerID	CompanyName	ContactName	ContactTitle	Address
1	ABC	ABC Company	John Smith	Owner	One My Way
2	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
3	ANATR	Ana Trujillo 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 a `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.

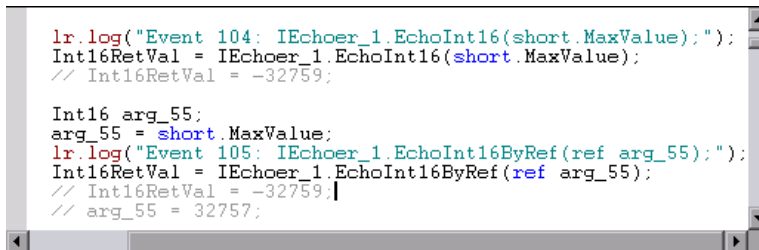
```
lr.message("The customer ID is ""+ lr.eval_string("{MyCustomerID}") + ");
```

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.

The above method is effective for ADO.NET environments. For primitive values, you should generate the script with output parameter values and examine the output parameters for correlations.

To correlate with output parameters:

- 1 Select **Tools > Recording Options**, and select the **General:Script** node.
- 2 Enable the **Insert output parameter values** option. Click **OK** to close Recording Options.
- 3 Select **Tools > Regenerate Script** to regenerate the script.
- 4 Search the commented output primitive values for correlations.



```
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**).

Configuring Application Security and Permissions

A Security Exception that occurs while recording an application is usually due to a lack of permissions—the recording machine does not have sufficient permissions to record the application. This is common where your application is not local, but on the Intranet or network.

To solve this problem, you need to allow the recording machine to access the application and the script with Full Trust.

One solution is to copy the application and save your script locally, since by default, users have Full Trust permissions to all local applications and folders.

An additional solution is to create new code groups that gives Full Trust to each application folder, and the script folder.

The procedure differs depending on whether you have Visual Studio installed on your machine.

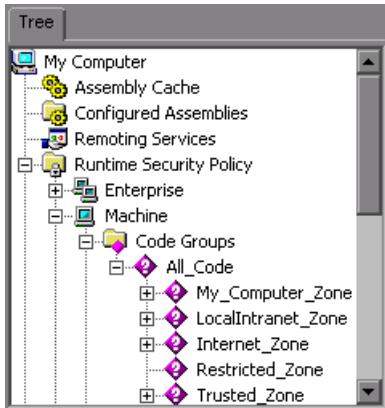
To grant Full Trust permissions to a specific folder if you do not have Visual Studio installed:

- 1** From the command prompt, run the caspol.exe application.
- 2** Set the desired permission.

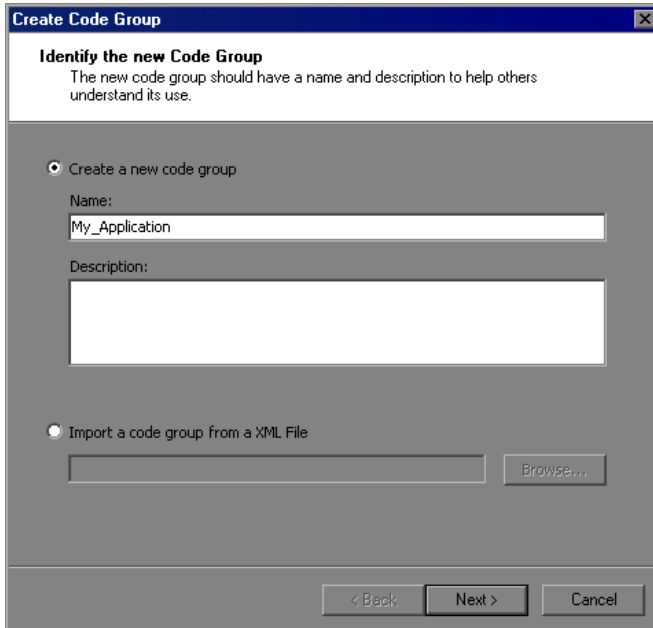
To grant Full Trust permissions to a specific folder if you have Visual Studio installed:

- 1** Open the .NET Configuration settings. Select **Start > Programs > Administrative Tools > Microsoft .NET Framework 2.0 Configuration**. The .NET Configuration window opens.

- 2 Expand the **Runtime Security Policy** node to show the Code Groups of the machine.



- 3 Select the **All_Code** node.
- 4 Select **Action > New ...**. The Create New Code Group dialog box opens.



- 5 Enter a name for a new Code Group for your application or script. Click **Next**.
- 6 Select the **URL** condition type. In the URL box, specify the full path of the application or script in the format `file://...` and click **Next**.

Create Code Group

Choose a condition type
The membership condition determines whether or not an assembly meets specific requirements to get the permissions associated with a code group.

Choose the condition type for this code group:
URL

The URL membership condition is true for all assemblies that originate from the URL specified below. Assemblies that meet this membership condition will be granted the permissions associated with this code group.

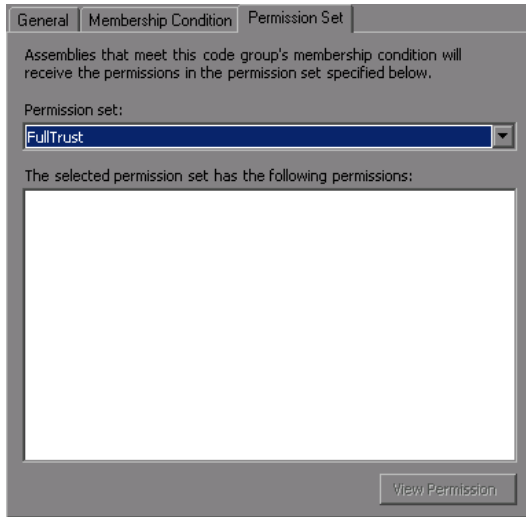
URL:
file://aut/ADO.NET.005/us/32/*

The URL must include the protocol such as 'ftp:/' or 'http:/'. An asterisk (*) can be used as a wildcard character at the end of the URL.

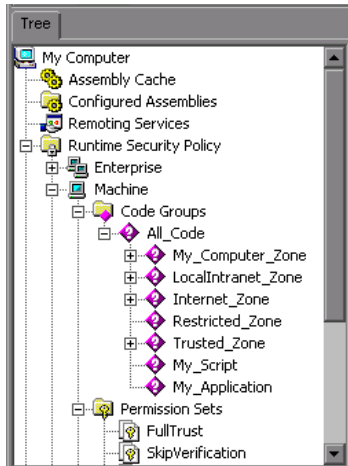
Examples:
http://www.microsoft.com/specific_assembly.dll
ftp://ftp.microsoft.com/pub/*

< Back Next > Cancel

7 Select the **FullTrust** permission set. Click **Next**.



8 Click **Finish** in the Completing the Wizard dialog box. The configuration tool adds your Code Group to the list of existing groups.



9 Repeat the above procedure for all .NET applications that you plan to record.

10 Repeat the above procedure for the Vuser script folder.

Note: Make sure that the script folder has **FullTrust** permissions on all Load Generator machines that are participating in the test (LoadRunner only).

Recording WCF Duplex Communication

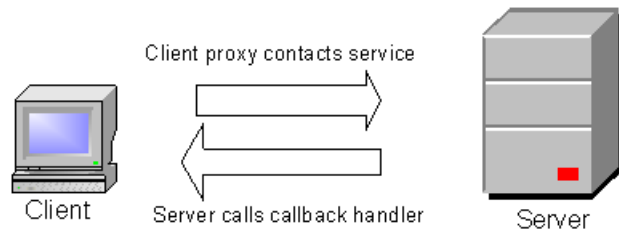
WCF (Windows Communication Foundation) is a programming model that unifies Web Services, .NET Remoting, Distributed Transactions, and Message Queues into a single Service-oriented programming model for distributed computing.

WCF creates a proxy object to provide data for the service. It also marshalls the data returned by the service into the form expected by the caller.

In addition to general support for the WCF environment, VuGen provides specialized support for applications that use WCF's duplex communication. In duplex communication, the client proxy contacts the service, and the service invokes the callback handler on the client machine. The callback handler implements a callback interface defined by the server. The server does not have to respond in a synchronous manner—it independently determines when to respond and invoke the callback handler.

The communication between the client and server is as follows:

- The server defines the service contract and an interface for the callback.
- The client implements the callback interface defined by the server.
- The server calls the callback handler in the client whenever needed.



When trying to record and replay duplex communication, you may encounter problems when the script calls the original callback methods. By default, the callback handlers are not included in the filter. You could customize the filter to include those callback handlers. However, the standard playback would be ineffective for a load test, since many of the callbacks are local operations such as GUI updates. For an effective load test you cannot replay the original callback method invoked by the server.

VuGen's solution is based on replacing the original callback handler with a dummy implementation. This implementation performs a typical set of actions that you can customize further for your application.

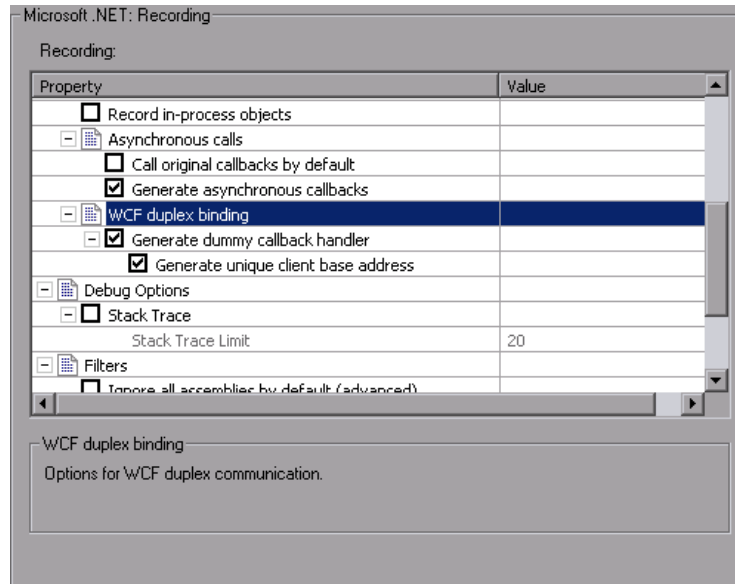
You instruct VuGen to replace the original callbacks by activating the **Generate Dummy Callback Handler** recording option. For more information, see below.

Setting WCF Recording Options

VuGen's recording options for WCF's duplex communications enable you to generate a script that will be effective for load testing. You can set recording options on the following areas:

- ▶ Generating Dummy Callback Implementations

► Recording Dual HTTP Bindings



Generating Dummy Callback Implementations

The **Generate Dummy Callback Handler** recording option instructs VuGen to replace the original callback in duplex communication with a dummy callback.

The dummy callback implementation performs the following actions:

- **Store arguments.** When the server calls the handler during replay, it saves the method arguments to a key-value in memory map.
- **Synchronize replay.** It stops the script execution until the next response arrives. VuGen places the synchronization at the point that the callback occurred during recording. This is represented in the script by a warning:

```
#warning: Code Generation Warning
// Wait here for the next response.
// The original callback during record was:...
```

As part of the synchronization, the script calls `GetNextResponse` to get the stored value.

```
Vuser<Callback_Name>.GetNextResponse();
```

Enabling the Dummy Callback Recording Option

By default, this option is enabled.

To enable this recording option:

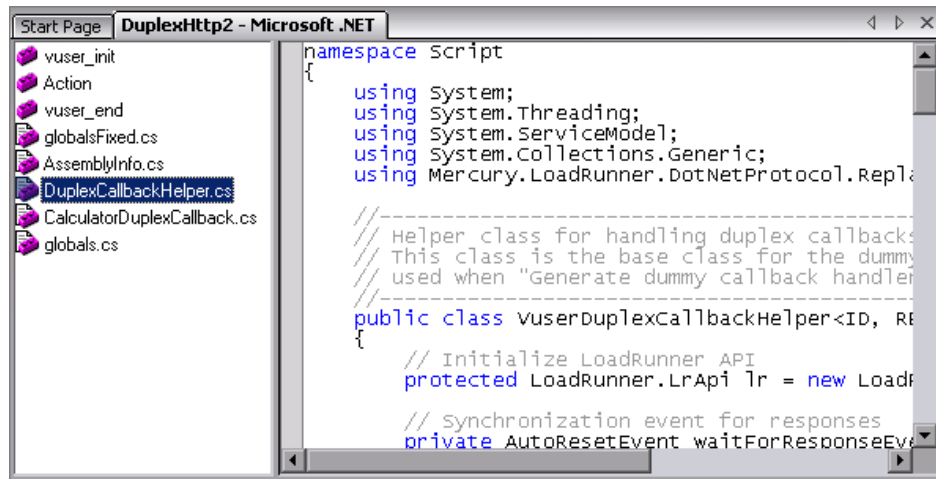
- 1 Select **Tools > Recording Options**.
- 2 Select the **Microsoft .NET:Recording** node.
- 3 Select **Generate Dummy Callback Handler**.

VuGen Implementation of a Duplex Callback

As part of the duplex communication solution, VuGen generates two support files:

- `DuplexCallbackHelper.<language>`
- `Callback_Name.<language>`

The following example shows the generated files for a Calculator application using duplex communication:



The Helper file serves as a general template for working with duplex callback handlers. It serves as a base class for the implementation of the callback.

The second file, **Callback_Name**, contains the implementation of the callback. The name of the callback implementation class is **Vuser<xxxx>** where *xxxx* is the name of the callback interface and it inherits from the **VuserDuplexCallbackHelper** class defined in the Helper file. VuGen creates separate implementation files for each interface.

This file performs two primary tasks:

- **Set Response.** It stores the data that came from the server in a map. It stores them with sequential IDs facilitating their retrieval. This method is called from the implementation of the callback interface. The following sample code demonstrates the dummy implementation of a callback method named **Result**. The method's arguments are stored in the map as an object array.

```
// -----
public virtual void Result(string operation, double result) {
    // Add here your own callback implementation and set the response data
    SetResponse(responseIndex++, new object[] {
        operation,
        result});
}
```

- **Get Response.** Waits for the next response to arrive. The implementation of `GetNextResponse` retrieves the next response stored in the map using a sequential indexer, or waits until the next response arrives.

The script calls `GetNextResponse` at the point that the original callback handler was called during recording. At that point, the script prints a warning:

```
// Wait here for the next response.
// The original callback during record was:
```

Replacement of the Callback in the Script

When you enable the **Dummy Callback** option (enabled by default), VuGen replaces the original duplex callback handlers with dummy implementations. The dummy implementation is called `Vuser <Callback Name>`. At the point of the original callback handler, the script prints a warning indicating that it was replaced.

Recording Dual HTTP Bindings

If your application employs dual HTTP Binding, since HTTP is inherently not a duplex protocol, the framework uses a standard port to receive response data being passed to the callback. When you attempt to run multiple instances of your application, you may be unable to do so using the same port number. VuGen provides you with an option of replacing the original client base address's port number with a unique port.

When you enable the **Generate Unique Client Base Address** recording option, VuGen checks the type of communication used by the application. If it detects dual HTTP communication, **WSDualHttpBinding**, it runs the **FindPort** utility (provided in `LrReplayUtils`) in the Helper file and finds unique ports for each instance of the callback.

This option is enabled by default. It is only relevant when you enable the above option, **Generate dummy callback handler**.

When you enable this option, VuGen generates the following code in the script:

```
#warning: Code Generation Warning
// Override the original client base address with a unique port number
DualProxyHelper.SetUniqueClientBaseAddress<XXXX>(YYYYYY);
```

Customizing the Dummy Implementation

You can modify the implementation file to reflect your environment.

Several suggested customization are:

- ▶ Timeouts

- Key Identifier
- Return Values
- Get Response Order
- Find Port

Timeouts

The default timeout for which the callback waits for the next response is 60000 msec, or one minute. To use a specific timeout, replace the call to **GetNextResponse** with the overloading method which gets the timeout as an argument as shown below. This method is implemented in the callback implementation file *<Callback_Name>* listed in the left pane after the **DuplexCallbackHelper** file.

```
// Get the next response.
// This method waits until receiving the response from the server
// or when the specified timeout is exceeded.
public virtual object GetNextResponse(int millisecondsTimeout) {
    return base.GetResponse(requestIndex++, millisecondsTimeout);
}
```

To change the default threshold for all callbacks, modify the **DuplexCallbackHelper** file.

```
// Default timeout threshold while waiting for response
protected int millisecondsTimeoutThreshold = 60000;
```

Key Identifier

Many applications assign key identifiers to the data, which connects the request and response to one another. This allows you to retrieve the data from the map using its ID instead of the built-in incremental index. To use a key identifier instead of the index, modify the file *<Callback_Name>* replacing the first base template parameter, **named ID**, with the type of your key identifier. For example, if your key identifier is a string you may change the first template argument from **int** to **string**:

```
public class VuserXXX : VuserDuplexCallbackHelper<string, object>
```

In addition, you may remove the implementation of `GetNextResponse()` and replace it with calls to `GetResponse(ID)` defined in the base class.

Return Values

By default, since VuGen supports *OneWay* communication, the implementation callback does not return any value or update an output parameter when it is invoked.

```
public virtual void Result(string operation, double result) {
    // Add here your own callback implementation and set the response data
}
```

If your application requires that the callback return a value, insert your implementation at that point.

Get Response Order

In VuGen's implementation, a blocking method waits for each response. This reflects the order of events as they occurred during recording—the server responded with data. You can modify this behavior to execute without waiting for a response or to implement the blocking only after the completion of the business process.

Find Port

The **FindPort** method in the Helper file is a useful utility that can be used in a variety of implementations. The Helper class uses this method to find unique ports for running multiple instances of the script. You can utilize this utility method for other custom implementations.

Recording Server Hosted By Client Applications

If the communication in your system is a server hosted by a client, VuGen's default solution for duplex communication will not be effective. In server hosted by client environments, it is not true duplex communication since the client opens the service and does not communicate through the Framework. For example, in queuing, the client sends a message to the service and opens a response queue to gather the responses.

To emulate a server hosted by a client, use the pattern depicted in the above solution—replace the original response queues with dummy callbacks and perform synchronization as required. For more information, contact HP support.

36

Microsoft .NET Filters

VuGen provides several built-in filters and also allows you to customize them.

This chapter includes:

- About Microsoft .NET Filters on page 533
- Guidelines for Setting Filters on page 535
- Setting a Recording Filter on page 539
- Working with the Filter Manager on page 541

About Microsoft .NET Filters

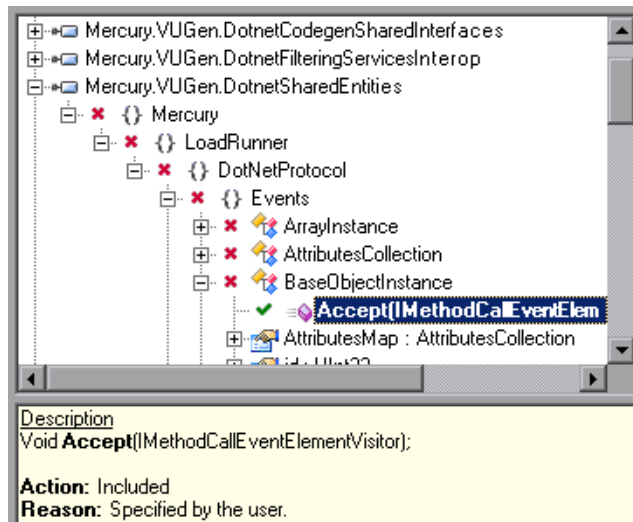
Recording filters indicate which assemblies, interfaces, namespaces, classes, or methods to include or exclude during the recording and script generation.

By default, VuGen provides built-in system filters for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation). These filters were designed to include the relevant interfaces for standard ADO.NET, Remoting, Enterprise Services, and WCF. VuGen also allows you to design custom filters.

Custom filters provide several benefits:

- ▶ **Remoting.** When working with .NET Remoting, it is important to include certain classes that allow you to record the arguments passed to the remote method.
- ▶ **Missing Objects.** If your recorded script did not record a specific object within your application, you can use a filter to include the missing interface, class or method.
- ▶ **Debugging.** If you receive an error, but you are unsure of it's origin, you can use filters to exclude methods, classes, or interfaces in order to pinpoint the problematic operation.
- ▶ **Maintainability.** You can record script in higher level, make script more easy to maintain and to correlate.

A filter manager lets you manipulate existing custom filters. It displays the assemblies, namespaces, classes, methods, and properties in a color-coded tree hierarchy.



The bottom pane provides a description of the assembly, namespace, class, method, property, or event. It also indicates whether or not it is included or excluded and a reason for the inclusion or exclusion.

The following sections describe when and how to customize a filter:

- Guidelines for Setting Filters
- Setting a Recording Filter
- Working with the Filter Manager

Guidelines for Setting Filters

When testing your .NET application, your goal is determining how the server reacts to requests from the client. When load testing, you want to see how the server responds to a load of many users.

When recording a .NET application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters, .NET Remoting, ADO.NET, Enterprise Services, and WCF, were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your .NET application's calls or exclude unnecessary calls. Using the Filter Manager, you can design custom filters to exclude the irrelevant calls and capture the server related calls.

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, you can use **Visual Studio** or a **Stack Trace** to help you determine which methods are being called by your application in order to include them in the filter. VuGen allows you to record with a stack trace that logs all of the methods that were called by your application.

Once you determine the required methods and classes, you include them using the Filter Manager. When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Tip: Strive to modify the filter so that your script will compile (Shift+F5) inside VuGen—a test with correct syntax. Then customize the filter further to create a functional script that runs inside VuGen.

Note that if you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this is that if you rerecord a script or regenerate the script, you will lose all of the manual changes.

Determining which Elements to Include or Exclude

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- ▶ **Top Down Approach.** An approach in which you include the relevant namespace and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined assembly which implements all client-server activity without involving any GUI elements, such as MyDataAccessLayer.dll.
- ▶ **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT assemblies and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- ▶ If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- ▶ If you identify a non-client/server call in your script, exclude its method in the filter.

- ▶ During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen **serializes** it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by **deserializing** it.
- ▶ VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the **LrReplayUtils.GetSerializedObject** method or, in WCF environments, **LrReplayUtils.GetSerializedDataContract**. VuGen stores serialized objects in the script's `\data\SerializedObjects` directory as XML files with indexes: **Serialization_1.xml**, **Serialization_2.xml** and so forth.
- ▶ When no rules are specified for a method, it is excluded by default. However, when the remoting environment is enabled, all remote calls are included by default, even if they are not explicitly included. To change the default behavior, you can add a custom rule to exclude specific calls which are targeted to the remote server.
- ▶ Arguments passed in remoting calls whose types are not included by the filter, are handled by the serialization mechanism. To prevent the arguments from being serialized, you can explicitly include such types in order to record the construction and the activity of the arguments.
- ▶ Exclude all activity which involves GUI elements.
- ▶ Add assemblies for utilities that may be required for the script to be compiled.

For information on how to include and exclude elements, see "Including and Excluding Elements" on page 545.

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:

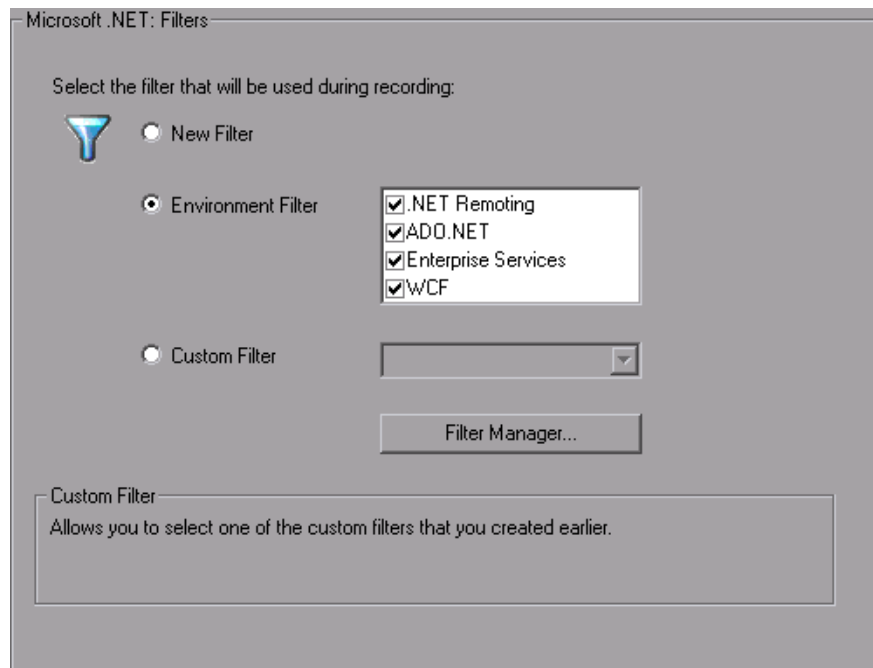
- 1** Create a new filter based on one of the built-in filters. If you know that the AUT (Application Under Test) does not use ADO.NET, Remoting, WCF, or Enterprise Services, clear that option since unnecessary filters may slow down the recording.
- 2** Set the **Stack Trace** option to true for both 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 536.
- 7** Record the application again. You should always rerecord the application after modifying the filter.
- 8** Repeat steps 4 through 7 until you get a simple script which can be easily maintained and correlated.
- 9** After creating an optimal script, turn off the **Stack Trace** options and regenerate the script. Open the Recording Options (CTRL+F7) and select the **Recording** node. Disable **Debug Options: Stack Trace** and **Code Generation: Show Stack Trace**. This will improve the performance of subsequent recordings.

- 10** Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information about the built-in correlation mechanism, see "Correlating Microsoft .NET Scripts" on page 516.

Setting a Recording Filter

The built-in filters, .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation), were designed to include the standard interfaces for those environments. For information on the benefits of filters, see "Guidelines for Setting Filters" on page 535.

When you decide to apply a filter to your recording, your first step is choosing an appropriate filter. You can use one or more of the environment filters or create a new one using the **Filters** recording options.



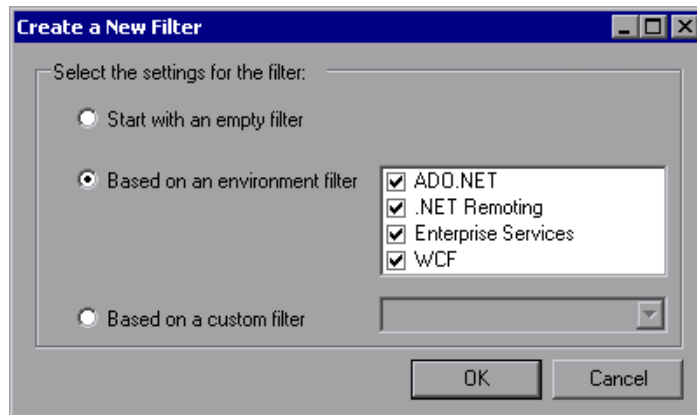
- **New Filter.** Indicates that you want to create a new filter.

- ▶ **Environment Filter.** Lists the available environment filters: .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation of Framework 3.0).
- ▶ **Custom Filter.** Shows the filters that you created earlier on the current machine.

After creating a filter, you can modify the properties of the filter using the Filter Manager. For more information, see "Including and Excluding Elements" on page 545.

To specify a filter:

- 1** Open the Filters recording option. Select **Tools > Recording Options** and select the **Microsoft .NET:Filters** node.
- 2** Select a filter option: **New Filter, Environment Filter, or Custom Filter.**
- 3** For a new filter, click **Create**. The Create a New Filter dialog box opens.



- 4** Select one of the filter options. To base your new filter on an environment filter, select the check box adjacent to one or more of the filters.
- 5** Click **OK**. The Filter Manager opens.

For existing filters, click on the **Filter Manager** button in the main recording options dialog box to open the Filter Manager.

Make the desired modifications to the filters and save the filter. For more information, see below.

Working with the Filter Manager

The Filter Manager lets you view and modify your filters, with the exception of Environment filters which can only be viewed—not modified as they are read-only.

You manage and manipulate your filters using the Filter Manager toolbar.



Managing Filters

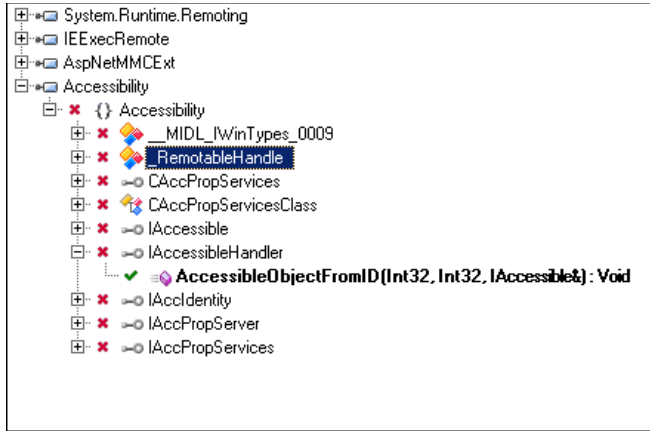
- ▶ **New.** Opens the Create a New Filter dialog box, in which you create an empty filter or a new filter based on an existing one
- ▶ **Save.** Saves the changes you made to filter.
- ▶ **Delete.** Deletes the selected custom filter. The Filter Manager prompts you for a confirmation.

Manipulating Filters

- ▶ **Add Reference.** Opens the Add Reference dialog box with a list of .NET Framework components or assemblies in the Public Assemblies folder. You can also browse the computer to locate a component that is not on the list. For more information, see "Adding References" on page 543.
- ▶ **Remove Reference.** Removes the assembly that is selected in the Filter Manager and all of the elements associated with it. The Filter Manager prompts you for a confirmation.
- ▶ **Include, Exclude, and Reset.** Includes or excludes an assembly, namespace, class, or method. You can also reset the inclusion or exclusion rule to its default state. For more information, see "Including and Excluding Elements" on page 545.
- ▶ **Back and Forward.** Navigates to the previous or next tree node visited by the user.
- ▶ **View Impact Log.** Opens the Impact log for the selected filter. The Impact log shows which nodes in the tree were affected by recent actions. For more information, see "Viewing an Impact Log" on page 547.









In addition, you can copy, paste, and rename filters using the standard Windows key combinations and right-click menu.








The Filter Manager tree uses symbols to illustrate the elements and their status:



- Element icons represent the type of element—assembly, namespace, class, method, structure, property, events, or interfaces.
- A check mark or X adjacent to the element icon, indicates whether or not the element is included or excluded.
- A bolded element indicates that it was explicitly included or excluded. This may be a result of being manually included or excluded by the user or by a pre-defined rule in the environment filter. If you reset a bolded node, it returns to its original, unbolded state.

The following table shows the icons that represent the various elements.

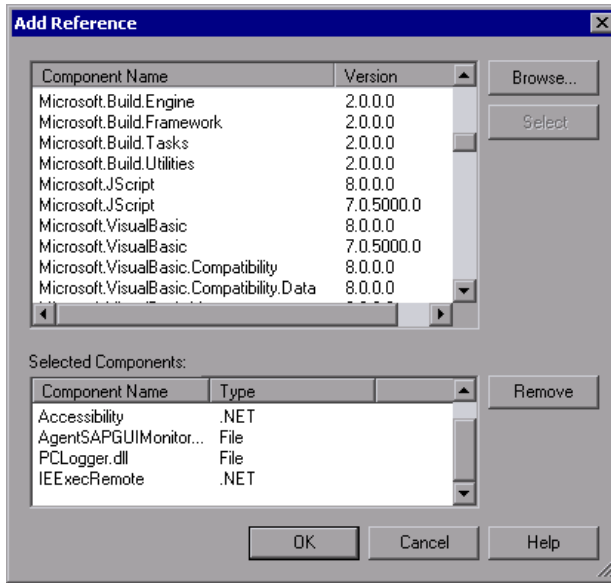
	assembly
	assembly that couldn't be loaded
	assembly that was partially loaded
	class
	constructor
	static constructor
	event
	static event

	interface
	method
	static method
	namespace
	property
	static property
	structure

Adding References

When you create a new filter, you must add assembly references to the filter to indicate its behavior. In addition, you can add references to pre-existing custom filters.

When you open the Add Reference dialog box, the Filter Manager lists all of the public assemblies in the GAC (Global Assembly Cache). You can add references that are not listed using the **Browse** button.



In the bottom pane, the **Type** column differentiates between the references. References that reside in the GAC are of the **.NET** type and references that do not reside in the GAC are of **File** type.

The list of references may include different versions of the same assembly. In this case, select the version that is most appropriate for your test.

To add a reference through the Filter Manager:

- 1** Click the **Add Reference** button on the toolbar. The Add Reference dialog box opens.
- 2** To add one of the listed items, select it and click **Select**. You can select multiple components using CTRL-CLICK. The bottom pane shows the selected references.
- 3** To add an assembly that is not in the list, click **Browse** and locate the reference on your file system or network.

- 4 Repeat the above steps for all of the references you want to add to your filter.
- 5 Review the list of references displayed in the bottom pane. To clear an item from the list, select it in the bottom pane and click **Remove**.
- 6 When you finish creating your list of references, click **OK** to close the dialog box and add the references to the filter. VuGen adds it to the end of the list of elements in the Filter Manager's tree. If any of the references you chose are not valid assemblies, VuGen will issue an error message.
- 7 To remove a reference from the Filter Manager's tree, select the parent assembly node and select **Remove Reference** from the right-click menu, or click the Remove Reference button on the toolbar.

After you add a reference, you should view it in the Filter Manager and determine if all the correct nodes are included or excluded. If necessary, you can include or exclude specific namespaces, classes, or methods. For more information, see below, Including and Excluding Elements.

Including and Excluding Elements

After you add references to the filter, you can view all of its nodes in the Filter Manager's tree. You can exclude specific namespaces, classes, or methods, or include those that were excluded by default or by other rules. The description in the Filter Manager's lower pane, indicates the reason for the inclusion or exclusion of the element.

The Filter Manager's toolbar provides the following buttons for including or excluding elements:



- **Include.** Indicated by a check mark, includes the selected element. If you manually include a parent node, the Filter Manager includes the child elements below it, provided that no other rule exists. For example, if you include a class, it will include all its methods unless you specifically excluded a method.



- ▶ **Exclude.** Indicated by an X, excludes the selected element. The child elements are also excluded unless they were included by another rule. By default, when you exclude a **class**, the Filter Manager applies the **Exclude** attribute to the class, but it allows the recording engine to record activity within the methods of the excluded class. When you exclude a **method**, however, the Filter Manager applies **Totally Exclude**, preventing the recording engine from recording any activity within the methods of the excluded class. Advanced users can modify these setting in the filter file. For more information, see "Advanced Information About Filter Files" on page 548.



- ▶ **Reset.** Removes the manual inclusion or exclusion rule. In this case, the element may be impacted by other parent elements.

The inclusion and exclusion rules have the following properties:

- ▶ The rules are hierarchical—if you add an include or exclude rule to a class, then the derived classes will follow the same rule unless otherwise specified.
- ▶ A rule on a class only affects its public methods, derived classes, and inner classes.
- ▶ A rule on a namespace affects all the classes and their public methods.
- ▶ Note that adding or removing assemblies does not necessarily affect the classes that they contain—you can remove an assembly, yet its methods may be recorded due to the hierarchical nature of the filter.
- ▶ As part of the filter design, several methods, such as **.ctor()** and **Dispose(bool)**, do not follow the standard hierarchal rules.

Note: The resetting of a parent node does not override a manual inclusion or exclusion applied to a child node. For example, if you manually **exclude** a method, and then reset its class, which by default **included** all sub-nodes, your method will remain excluded.

Properties and events are view-only and cannot be included or excluded through the Filter Manager. In addition, several system related elements are protected and may not be altered.

For tips about including and excluding elements in the filter, see "Determining which Elements to Include or Exclude" on page 536.

To add or remove assemblies, use the **Add** and **Remove** Reference buttons as described in "Adding References" on page 543. The following section describes how to include namespaces, classes, and methods.

To include or exclude an element:

- 1** Expand the tree hierarchy and select a namespace, class, or method.
- 2** To include an element, select it and click the **Include** button or use the Include command on the right-click menu.
- 3** To exclude an element, click the **Exclude** button or use the Exclude command on the right-click menu.
- 4** To reset an element to its default settings, click the **Reset** button or select **Reset** from the right-click menu.

To verify that the change took effect, select the component and view the bottom pane.

Description

Int32 **ExecuteAsAssembly**(String, Evidence, Byte[], AssemblyHashAlgorithm);

Action: Included

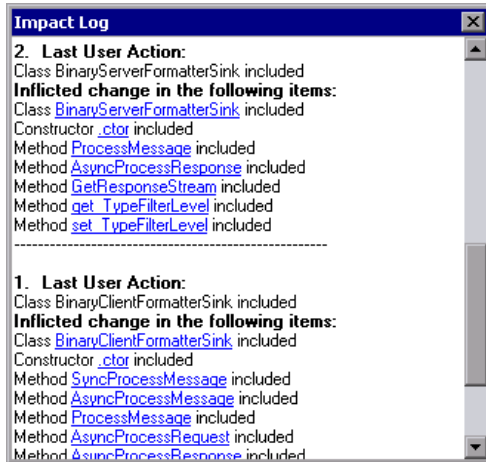
Reason: Specified by the user.

Viewing an Impact Log

The Impact Log indicates what your last changes were and how they affected your filter. The user actions are listed in descending order, with the latest changes at the top.

For each element affected by your manual inclusion or exclusion, the log indicates how it affected the element. It also provides a link to that element in the Filter Manager.

To view the Impact Log, click the Impact Log button on the Filter Manager's toolbar or select **Actions > View Impact Log** in the Filter Manager window.



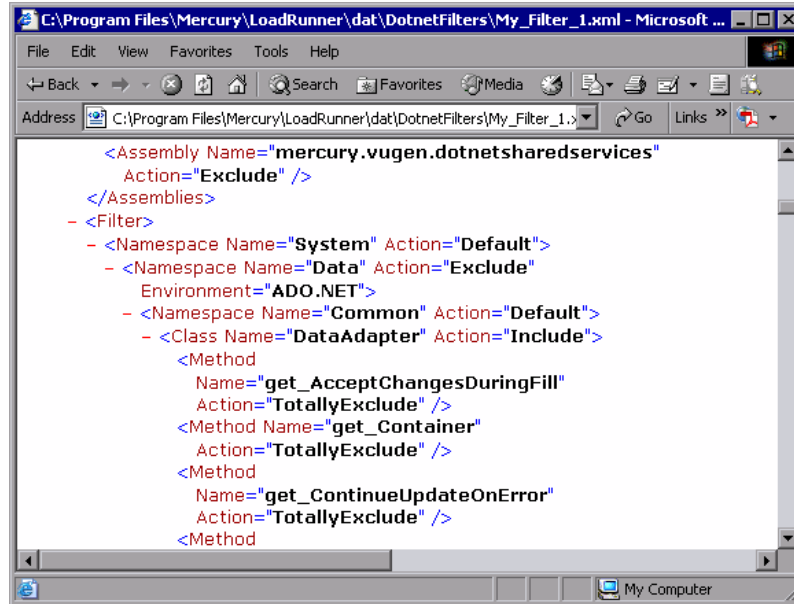
Advanced Information About Filter Files

In the Filter Manager's tree hierarchy, it only displays public classes and methods. It does not show non-public classes or delegates.

You can add classes or methods that are not public by manually entering them in the filter's definition file.

The filter definition files, `<filter_name>.xml` reside in the `dat\DotnetFilters` folder of your installation. The available Action properties for each element are: **Include**, **Exclude**, or **Totally Exclude**. For more information, see "Including and Excluding Elements" on page 545.

By default, when you exclude a **class**, the Filter Manager applies **Exclude**, excluding the class, but including activity generated by the excluded class. When you exclude a **method**, however, it applies **Totally Exclude**, excluding all referenced methods.



For example, suppose Function A calls function B. If Function A is **Excluded**, then when the service calls Function A, the script will include a call to Function B. However, if function A is **Totally Excluded**, the script will not include a call to Function B. Function B would only be recorded if called directly—not through Function A.

VuGen saves a backup copy of the filter as it was configured during the recording, **RecordingFilterFile.xml**, in the script's **data** folder. This is useful if you made changes to the filter since your last recording and you need to reconstruct the environment.

37

Web (HTTP/HTML, Click and Script) Protocols

You use VuGen to develop Web Vuser scripts based on your actions while you operate a client browser.

This chapter includes:

- ▶ About Developing Web Level Vuser Scripts on page 551
- ▶ Introducing Web Vusers on page 552
- ▶ Understanding Web Vuser Technology on page 553
- ▶ Selecting a Web Vuser Type on page 553
- ▶ Getting Started with Web Vuser Scripts on page 557
- ▶ Recording a Web Session on page 559
- ▶ Converting Web Vuser Scripts into Java on page 560
- ▶ Support for Push Technology on page 561

About Developing Web Level Vuser Scripts

You use VuGen to develop Web Vuser scripts. While you navigate through a site performing typical user activities, VuGen records your actions and generates a Vuser script. When you run the script, the resulting Vuser emulates a user accessing the Internet.

After you create a Vuser script, you run the script in stand-alone mode using VuGen. When the execution is successful, you are ready to integrate the Vuser script into a scenario. For details on how to integrate a Vuser script into a scenario, see the *HP LoadRunner Controller User's Guide*.

For certain Vuser types, you can create a Business Process Report for Microsoft Word that provides information about the script and the events that were recorded. For more information, see "Creating Business Process Reports" in *Volume I-Using VuGen*.

Introducing Web Vusers

Suppose you have a Web site that displays product information for your company. The site is accessed by potential customers. You want to make sure that the response time for any customer query is less than a specified value (for example, 20 seconds)—even when a large number of users (for example, 200) access the site simultaneously. You use Vusers to emulate this case, where the Web server services simultaneous requests for information. Each Vuser could do the following:

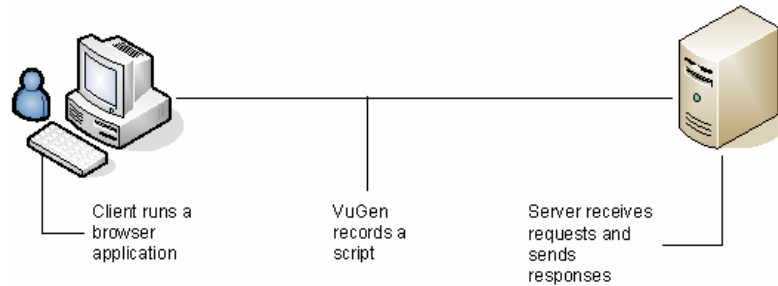
- ▶ Load a home page
- ▶ Navigate to the page containing the product information
- ▶ Submit a query
- ▶ Wait for a response from the server

You can distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

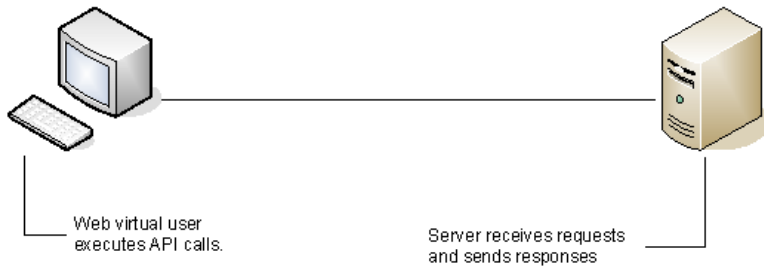
The program that contains the calls to the server API is called a Vuser script. It emulates a browser application and all of the actions performed by the browser. Using the Controller, you assign the script to multiple Vusers. The Vusers execute the script and emulate user load on the Web server.

Understanding Web Vuser Technology

VuGen creates Web Vuser scripts by recording the activity between a browser and a Web server. VuGen monitors the client (browser) end of the system and traces all the requests sent to, and received from, the server.



When you run a recorded Vuser script, the Vuser communicates directly with the server without relying on client software. Instead, the Vuser script executes calls directly to the Web server via API functions.



Selecting a Web Vuser Type

When creating a new Web Vuser script, you can select one of the following types of Web Vusers:

- Web (Click and Script)
- Web (HTTP/HTML)

Web (Click and Script)

The Web (Click and Script) Vuser is a solution for recording Web sessions on a user-action GUI level. VuGen creates a GUI-level script that intuitively represents actions in the Web interface. For example, it generates a **web_button** function when you click a button to submit information, and generates a **web_edit_field** function when you enter text into an edit box.

Web (Click and Script) Vusers support non-HTML code such as Javascript on the client side. VuGen creates an intuitive script that accurately emulates your actions on the Web page and executes the necessary Javascript code.

Web (Click and Script) Vusers handle most correlations automatically, reducing the scripting time. In most cases, you do not need to define rules for correlations or perform manual correlations after the recording.

Web (Click and Script) Vusers also allow you to generate detailed Business Process Reports which summarize the script.

For example, when you click a button to submit data, VuGen generates **web_button**. If the button is an image, VuGen generates **web_image_submit**. In the following example, a user clicked the login button.

```
...
web_image_submit("Login",
    "Snapshot=t4.inf",
    DESCRIPTION,
    "Alt=Login",
    "Name=login",
    "FrameName=navbar",
    ACTION,
    "ClickCoordinates=31,6",
    LAST);}
```

The next section illustrates a user navigating to the Asset ExpressAdd process under the Manage Assets branch. The user navigates by clicking the text links of the desired branches, generating `web_text_link` functions.

```
web_text_link("Manage Assets_2",
  DESCRIPTION,
  "Text=Manage Assets",
  "Ordinal=2",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);

web_text_link("Use",
  DESCRIPTION,
  "Text=Use",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);

web_text_link("Asset ExpressAdd",
  DESCRIPTION,
  "Text=Asset ExpressAdd",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);
```

In the following example, `web_list` emulates the selection of a list item.

```
...
web_list("Year",
  DESCRIPTION,
  "Name=Year",
  "FrameName=CalFrame",
  ACTION,
  "Select=2000",
  LAST);
```

When you click on an image that is associated with an image map, VuGen generates a `web_map_area` function.

```
web_map_area("map2_2",  
            DESCRIPTION,  
            "MapName=map2",  
            "Ordinal=20",  
            "FrameName=CalFrame",  
            ACTION,  
            "UserAction=Click",  
            LAST);
```

Note: Web (Click and Script) Vusers do not support Applets or VB Script. If the Web site under test contains these items, use the Web (HTTP/HTML) user.

Web (HTTP/HTML)

When recording a Web (HTTP/HTML) script, VuGen records the HTTP traffic between the browser and the server. The scripts contain detailed information about the recorded traffic.

The Web (HTTP/HTML) Vuser provides two recording levels: **HTML-based script** and **URL-based script**. These levels 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 Chapter 20, "Setting Recording Options for Web Vusers" in *Volume I-Using VuGen*.

Tip: For most applications, including those with JavaScript, use Web (Click and Script) Vusers. For browser applications with applets and VB Script or for non-browser applications, use the Web (HTTP/HTML) Vuser.

Getting Started with Web Vuser Scripts

This section provides an overview of the process of developing Web Vuser scripts.

To develop a Web Vuser script:

1 Create a new script using VuGen.

Click on VuGen's **Start Page** tab and click on **File > New**. Select Web (Click and Script) or Web (HTTP/HTML) Vuser script from the **e-business** category, in either single or multiple protocol mode.

For details about creating a new script, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Set the recording options.

Set the recording options. For information about setting common Internet recording options, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

For details about selecting a recording level, see Chapter 20, "Setting Recording Options for Web Vusers" in *Volume I-Using VuGen*.

For Web (Click and Script) specific options, see Chapter 17, "Click and Script Recording" in *Volume I-Using VuGen*.

3 Record a browser session.

Record your actions while you navigate your Web site.

For details about creating a new script, see "Recording with VuGen" in *Volume I-Using VuGen*.

4 Enhance the recorded Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, checks, and service steps.

For details, see Chapter 40, "Web (HTTP/HTML, Click and Script) Text and Image Verification", Chapter 41, "Modifying Web and Wireless Vuser Scripts", and Chapter 42, "Web (HTTP/HTML) Correlation Rules."

5 Define parameters (optional).

Define parameters for the fixed values recorded into your script. By substituting fixed values with parameters, you can repeat the same Vuser action many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

6 Configure the run-time settings.

The run-time settings control Vuser behavior during script execution. These settings include general run-time settings (iteration, log, think time, and general information), and Web-related settings (proxy, network, and HTTP details).

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

7 Perform correlation.

For Web (HTTP/HTML) scripts, Scan your Vuser script for correlations and use one of VuGen's mechanisms to implement them.

For details on setting up automatic correlation, see Chapter 42, "Web (HTTP/HTML) Correlation Rules." For information on correlation after the recording, see Chapter 43, "Web (HTTP/HTML) -Correlation After Recording."

8 Run and debug the Vuser script using VuGen.

Run the Vuser script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen* and "Viewing Test Results" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Recording a Web Session

When you record a Web session, VuGen monitors all the actions that you perform in your Web browser. Your activities can include hyperlink jumps (both hypertext and hypergraphic) and form submissions. While recording, VuGen saves the recorded actions in a Web Vuser script.

Each Vuser script that you create contains at least three sections: **vuser_init**, one or more **Actions**, and **vuser_end**. During recording, you can select the section of the script into which VuGen will insert the recorded functions. The `vuser_init` and `vuser_end` sections are generally used for recording server logon and logoff procedures, which are not repeated when you run a Vuser script with multiple iterations.

You should therefore record a Web session into the Actions sections so that the complete browser session is repeated for each iteration.

Converting Web Vuser Scripts into Java

VuGen provides a utility that enables you to convert a script created for a Web Vuser into a script for Java Vusers. This also allows you to create a hybrid Vuser script for both Web and Java.

To convert a Web Vuser script into a Java Vuser script:

- 1** Create an empty Java Vuser script and save it.
- 2** Create an empty Web Vuser script and save it.
- 3** Record a Web session using standard HTML/HTTP recording.
- 4** Replay the Web Vuser script. When it replays correctly, cut and paste the entire script into a text document and save it as a text **.txt** file. In the text file, modify any parameter braces from the Web type, "{ }" to the Java type, "< >".
- 5** Open a DOS command window and go to your product's **dat** directory.

6 Type the following command:

```
<application_directory>\bin\sed -f web_to_java.sed filename > outputfilename
```

where **filename** is the full path and filename of the text file you saved earlier, and **outputfilename** is the full path and filename of the output file.

- 7** Open the output file, and copy its contents into your Java Vuser script action section at the desired location. If you are pasting the contents into an empty custom Java template (Java Vuser type), modify the line containing `public int action()` as follows:

```
public int action() throws Throwable
```

This change is done automatically for recorded Java users (RMI and CORBA).

Parameterize and correlate the Vuser script as you would with an ordinary Java script and run it.

Support for Push Technology

Vuser scripts run steps sequentially, one step cannot start until the previous step has completed. A step does not complete until all requested data has finished downloading. Some websites use push technology in which downloads are not meant to complete. Data is periodically downloaded as updates occur.

When a Vuser script contains steps which access resources using push technology, the Vuser script gets stuck on the step until a timeout occurs. This is a “false” timeout because the site is functioning as designed.

Expert users can work around this problem by changing the conditions for the step to complete. For more information, see the *Online Function Reference* (**Help > Function Reference**).

38

Web (Click and Script) Tips

The Record, Replay and Enhancement tips provide guidelines for developing Web (Click and Script) scripts.

This chapter includes:

- ▶ Recording Issues on page 563
- ▶ Recording Tips on page 565
- ▶ Replay Problems on page 567
- ▶ Replay Tips on page 569
- ▶ Miscellaneous Problems on page 570
- ▶ Miscellaneous Tips on page 572
- ▶ Enhancing Your Web (Click and Script) Vuser Script on page 573

The following information applies to the Web (Click and Script), AJAX (Click and Script), SAP (Click and Script), Oracle Web Applications 11i, and PeopleSoft Enterprise protocols.

Recording Issues

The following section lists the most common recording problems:

Firefox is not supported

Only Internet Explorer is supported for Web (Click and Script). To record browser activity on Firefox, use the Web (HTTP/HTML) protocol.

Application behaves differently while being recorded

If your application behaves differently during recording, than it does without recording, you should determine if the recording problem is unique to Web (Click and Script). The effect may be that a Web page will not load, part of the content is missing, a popup window does not open, and so forth.

Create a new Web (HTTP/HTML) script and repeat the recording.

In the event that the recording fails in Web (HTTP/HTML), we recommend that you disable socket level recording (see "Disable socket level recording" on page 566).

The problem may be the result of an event listener. Use trial and error to disable event listeners in the **Web Event Configuration** Recording Options, and then re-record your session as a Web (Click and Script) user.

To disable an event listener:

- ▶ Open the Recording Options. Select **Tools > Recording Options** and select the **GUI Properties:Web Event Configuration** node.
- ▶ Click **Custom Settings** and expand the **Web Objects** node. Select an object.
- ▶ Select **Disabled** from the list in the **Record** column for the relevant Web object. If the recording still does not work, enable the listener you previously disabled, and try disabling another one. Repeat these steps until your recording succeeds.

Dynamic menu navigation was not recorded

A dynamic menu is a menu that dynamically changes depending on where you select it. If the dynamic menu navigation was not recorded, record again using "high" event configuration mode.

To set the configuration level to high:

- ▶ Open the Recording Options. Select **Tools > Recording Options** and select the **GUI Properties:Web Event Configuration** node.
- ▶ Move the slider to **High**.

Certain user actions were not recorded

Check if there is a Java applet running inside the browser. If not, record the script with the Web (HTTP/HTML) protocol.

Recording Tips

Use the mouse and not the keyboard

It is preferable to click on an object with the mouse rather 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 564.

Disable socket level recording

In certain cases, the capturing of the socket level messages disrupts the application. For most recordings, socket level data is not required. To prevent the recording of socket level data, disable the option in the recording options. For more information, see the section about recording with Click and Script.

Enable the record rendering-related property values

If the client-side scripts of the application use a lot of styling activities, enable the **Record rendering-related property values** option before recording the script. For example, enable this option to record additional DOM properties such as **offsetTop**. Note that enabling this option may decrease the recording speed.

To enable record rendering-related property values:

- Open the Recording Options. Select **Tools > Recording Options** and select the **GUI Properties:Advanced** node.

Replay Problems

GUI object not found

Does the error occur at the beginning of the second iteration?

If the error occurs at the beginning of the second iteration's Action section, it is probably the result of a starting page that was present for the first iteration, but missing for the second one. If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page, so that the same link will be visible at the end of the business process.

Is it a text link containing non-ASCII characters?

If the problem occurs with non-ASCII characters, you should instruct VuGen to 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 Tips

The following tips may help you in troubleshooting your problems:

Do not reorder

Do not reorder the statements within a recorded script. Also, copying segments of code from one Action to another is not recommended.

Convert non-ASCII characters

If your links contain non-ASCII characters, you should instruct VuGen to convert the data to or from the UTF-8 format.

To enable UTF-8 conversion:

- Open the Recording Options. Select **Vuser > Run-Time Settings** and select the **Internet Protocol:Preferences** node.
- Click **Options** to open the Advanced Options dialog box.
- Locate the **Convert from/to UTF-8** option and set it to **Yes**.

Alternatively, view the list of alternatives that are displayed when a link is not found. Enter the displayed text as-is, such as the hex escape sequences `\xA0` or any other non-standard format.

Run same sequence of actions twice

In some cases, you can only perform a certain process once, such as deleting a user from the database. Replay will fail after the first iteration, because the action 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 Problems

Out of memory error in JavaScript

Increase the JavaScript memory in the Run-Time settings.

To increase the JavaScript memory size:

- 1 Open the Recording Options. Select **User > Run-Time Settings** and select the **Internet Protocol:Preferences** node.
- 2 Click **Options** to open the Advanced Options dialog box.
- 3 Locate the **Memory Management JavaScript Runtime Memory Size (Kb)** and **Memory Management JavaScript Stack Memory Size (Kb)** options.
- 4 Increase the memory sizes to 512 or higher.

VuGen displays JavaScript errors

If VuGen displays JavaScript errors in the Replay Log, verify that the Javascript itself does not contain errors, by enabling IE (Internet Explorer) script errors.

To show script errors:

- 1** Open Internet Explorer. Select **Tools > Internet Options** and select the **Advanced** tab.
- 2** Enable the **Display a notification about every script error** under the **Browsing** section.
- 3** Rerun the application in IE. If IE displays script errors, then there is a problem with the JavaScript application. If it is not possible to fix the application, you can safely ignore the corresponding replay errors.

Problems following parameterization

If you encounter problems only after you have parameterized values, verify that the values are valid for your application. Perform business process with the value of the parameter and verify that the application accepts it.

Problems with applications that utilize styling actions

If the client-side scripts of the application use a lot of styling activities, you should record the script again after enabling the **Record rendering-related property values** option. This enables the recording of additional DOM objects.

To enable record rendering-related property values:

- 1** Open the Recording Options. Select **Tools > Recording Options** and select the **GUI Properties:Advanced** node.
- 2** Enable the **Record rendering-related property values** option. Re-record the script.

Miscellaneous Tips

The following additional tips may help you in troubleshooting your problems:

Search for warnings

Search for warnings or alerts in the Replay Log.

Verify the response

Verify the response of the previous step is correct using `web_reg_find`. For more information, see the *Online Function Reference* (**Help > Function Reference**).

Use alternate navigation

For problematic steps or those using Java applets, Use **Alternative Navigation** to replace the Web (Click and Script) step with an HTTP level step. Note that the HTTP level steps may require manual correlations. To perform Alternative Navigation, select a step in Tree View, or the text in Script View, and select **Replace with alternative navigation** from the right-click menu.

Working with the Kerberos Protocol

If you are using the Kerberos Protocol for authentication, you must customize VuGen to properly convene authorization sessions. Advanced users can attempt to perform this customization themselves.

In order for the Kerberos Protocol to work properly, create a `krb5.ini` file and put it in an available directory. Save the full pathname of `krb5.ini` into the `KRB5_CONFIG` 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.

Enhancing Your Web (Click and Script) Vuser Script

The following section describes several enhancements that can assist you in creating your script.

Most of the features described below are enhancements to the API functions. For detailed information about the functions and their arguments, see the *Online Function Reference* (**Help > Function Reference**) or click F1 on any function.

Adding conditional steps

The Web (Click and Script) functions, **web_xxxx**, allow you to specify conditional actions during replay. Conditions are useful, for example, if you need to check for an element and perform an action only if the element is found.

For example, suppose you perform an Internet search and you want to navigate to all of the result pages by clicking Next. Since you do not know how many result pages there will be, you need to check if there is a Next button, indicating another page, without failing the step. The following code adds a verification step with a notification—if it finds the Next button, it clicks on it.

```
While (web_text_link("Next",
DESCRIPTION,
    "Text=Next",
    VERIFICATION,
    "NotFound=Notify",
    ACTION,
    "UserAction=Click",
    LAST) == LR_PASS);
```

For details about the syntax and use of the VERIFICATION section, see the *Online Function Reference* (**Help > Function Reference**).

Checking a page title

In `web_browser` steps, you can use the title verification recording option to make sure that the correct page is downloaded. You can instruct the Vuser to perform this check automatically for every step or every navigation to a new top level window.

In addition, you can manually add title verifications to your script at the desired locations, using both exact and regular expression matches.

```
web_browser("test_step",  
DESCRIPTION,  
...  
VERIFICATION,  
  "BrowserTitle=Title",  
  ACTION,]  
,  
LAST);
```

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

You can set title verification options directly from within the Recording options. For more information, see 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, Vuusers 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`.

39

Web (HTTP/HTML, Click and Script) Functions

VuGen helps you create Web Vuser scripts that describe user actions on Web sites. Each script contains functions that correspond directly to each of the actions taken.

This chapter includes:

- ▶ About Web Vuser Functions on page 580
- ▶ Adding and Editing Functions on page 581
- ▶ General Web (Click and Script) API Notes on page 583
- ▶ Using Cache Data on page 585

The following information applies to Web (Click and Script), Web (HTTP/HTML), Oracle Web Applications 11i, and PeopleSoft Enterprise.

About Web Vuser Functions

The functions developed to emulate Internet communication between a browser and a Web server are called Web Vuser functions. Each Web Vuser function has a **web** prefix. Some functions are generated when you record a script; others you must manually insert into the script.

The Web functions are categorized as follows:

- ▶ Action Functions
- ▶ Authentication Functions
- ▶ Check Functions
- ▶ Connection Definition Functions
- ▶ Concurrent Group Functions
- ▶ Cookie Functions
- ▶ Correlation Functions
- ▶ Filter Functions
- ▶ Header Functions
- ▶ Proxy Server Functions
- ▶ Replay Functions
- ▶ Miscellaneous Functions

Web (Click and Script) Vusers use other functions to emulate user actions.

Most functions which are not Action functions, may be used in Web (Click and Script) Vuser scripts. However, the **web_concurrent_start** and **web_concurrent_end** functions are specific to Web (HTTP/HTML) Vuser scripts.

For detailed information and examples of the Web Protocol functions, see the *Online Function Reference* (**Help > Function Reference**).

For more information on adding general Vuser functions to scripts, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

Adding and Editing Functions

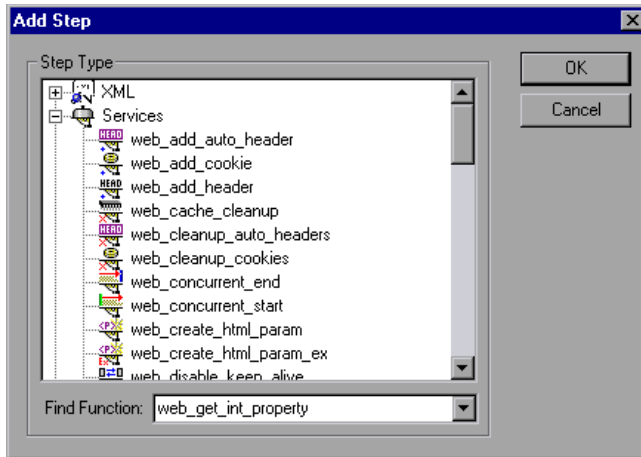
Many of the Web Vuser functions are created during the recording session. You can also manually add and edit Web Vuser functions after recording in both the Tree view and Script view.

When you select a new step to add to your script, VuGen categorizes the steps in the following types:

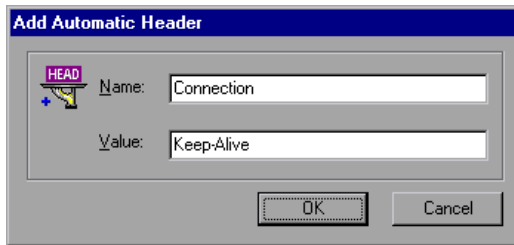
Step Type	Description
Service	A Service step is a function that does not make any changes in the Web application context. Rather, service steps perform customization tasks such as setting proxies, providing authorization information, and issuing customized headers.
URL	A URL step is generated when you type a URL into the browser or use a bookmark to access a specific Web page. Each URL icon represents a web_url function in the Vuser script. The default label of a URL icon is the last part of the URL of the target page.
Link	VuGen adds a Link step when you click a hypertext link while recording. Each Link step represents a web_link function in the Vuser script. The default label of the step is the text string of the hypertext link (only recorded for the HTML-based recording level).
Image	VuGen adds an Image step to the Vuser script when you click a hypergraphic link during recording. Each Image step represents a web_image function in the Vuser script. If the image in the HTML code has an ALT attribute, then this attribute is used as the default label of the icon. If the image in the HTML code does not have an ALT attribute, then the last part of the SRC attribute is used as the icon's label (only recorded for the HTML-based recording level).
Submit Form / Submit Data	VuGen adds a Submit Form or Submit Data step when you submit a form while recording. The default label of the step is the name of the executable program used to process the form (Submit Form only recorded for the HTML-based recording level).
Custom Request	VuGen adds a Custom Request step to a Vuser script when you record an action that VuGen cannot recognize as any of the standard actions (i.e., URL, link, image, or form submission). This is applicable to non-standard HTTP applications.

To add a new function to an existing Vuser script:

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



- 2 Select the desired function and click **OK**. Most Web Vuser functions are under the **Services** category. The Properties dialog box for that function opens. This dialog box lets you specify the function's arguments.



- 3 Specify the properties and click **OK**. VuGen inserts the function with its arguments at the location of the cursor.

You can edit existing steps by opening the Properties dialog box and modifying the argument values. This is only valid for protocols that support tree view (not available for WAP).

To edit an existing step:

- 1** In the tree view, select **Properties** from the right-click menu. The Properties dialog box for that function opens.
- 2** Modify the argument values as necessary and click **OK**.

You can manually add general Vuser functions such as transactions, rendezvous, comments, and log functions during recording. For more information, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

General Web (Click and Script) API Notes

This section lists general notes about the Web (Click and Script) functions. Note that you can specify a regular expression for most object descriptions, by preceding the text with "/RE" before the equals sign. See the Function Reference (Help > Function Reference) for more details. For example:

```
web_text_link("Manage Assets",  
DESCRIPTION,  
"Text/RE=(Manage Assets)|(Configure Assets)",  
ACTION,  
"UserAction=Click",  
LAST);
```

Ordinals

The Ordinal attribute is a one-based index to distinguish between multiple occurrences of objects with identical descriptions. In the following example, the two recorded `web_text_link` functions have identical arguments, except for the ordinal. The ordinal value of 2, indicates the second occurrence.

```
web_text_link("Manage Assets",
  DESCRIPTION,
  "Text=Manage Assets",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);

web_text_link("Manage Assets_2",
  DESCRIPTION,
  "Text=Manage Assets",
  "Ordinal=2",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);
```

Empty Strings

There is a difference between not specifying an argument and specifying it as an empty string. When you do not specify an argument, VuGen uses the default value or ignores it. When you list an argument, but assign it an empty string as a value, VuGen attempts to find a match with an empty string or no string at all. For example, omitting the id argument instructs VuGen to ignore the id property of the HTML element. Specifying "ID=" searches for HTML elements with no id property or with an empty ID.

```
web_text_link("Manage Assets_2",
  DESCRIPTION,
  "Text=Manage Assets",
  "Id=",
  "FrameName=main",
  ACTION,
  "UserAction=Click",
  LAST);
```


Using Cache Data

You can save stored data into your browser's cache, and load it at a later point in the script.

To implement this within your script, you manually add the `web_dump_cache` and `web_load_cache` functions.

Dumping Information to the Cache

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 `Vuser` 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 `Vuser` name value:

```
Ku001paycheck.cache
Ku002paycheck.cache
Ku003paycheck.cache
...
```

You can modify the first and second arguments (`paycheckcache` and `paycheck` in this example) to reflect the current transaction name. Place this function at the end of your script, after you have loaded all of the resources.

Loading Information from the Cache

The `web_load_cache` function loads a cache file whose location is specified in the `FileName` argument. Note that the `web_load_cache` function requires the cache file to exist. Therefore, you can only run this function after running `web_dump_cache`.

In the following example, the **web_load_cache** function loads the **paycheck** cache files from **C:\temp**.

```
web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",LAST)
```

Inserting the Caching Functions

The following procedure describes how to use caching functions.

To use the caching functions:

- 1** Insert the **web_dump_cache** function into your script.
- 2** Run the script at least once.
- 3** Insert the **web_load_cache** function into your script before the Vuser actions.
- 4** Comment out the **web_dump_cache** function.
- 5** Run and save the script.

Caching Example

The following example illustrates a PeopleSoft Enterprise Vuser viewing the details of his paycheck.

```
Action()
{
// web_add_cookie("storedCookieCheck=true; domain=pbntas05; path=");

web_load_cache("ActionLoad", "FileName=c:\\temp\\{VuserName}paycheck", LAST);

    web_browser("signon.html",
        DESCRIPTION,
        ACTION,
        "Navigate=http://pbntas05:8200/ps/signon.html",
        LAST);
    lr_think_time(35);

    web_edit_field("userid",
        "Snapshot=t1.inf",
        DESCRIPTION,
        "Type=text",
        "Name=userid",
        ACTION,
        "SetValue={VuserName}",
        LAST);
```

```
web_edit_field("pwd",
    "Snapshot=t2.inf",
    DESCRIPTION,
    "Type=password",
    "Name=pwd",
    ACTION,
    "SetValue=HCRUSA_KU0007",
    LAST);

lr_start_transaction("login");
    web_button("Sign In",
        "Snapshot=t3.inf",
        DESCRIPTION,
        "Type=submit",
        "Tag=INPUT",
        "Value=Sign In",
        LAST);
lr_end_transaction("login", LR_AUTO);

web_image_link("CO_EMPLOYEE_SELF_SERVICE",
    "Snapshot=t4.inf",
    DESCRIPTION,
    "Alt=",
    "Name=CO_EMPLOYEE_SELF_SERVICE",
    "Ordinal=1",
    ACTION,
    "ClickCoordinate=10,10",
    LAST); ...

web_text_link("Sign out",
    "Snapshot=t7.inf",
    DESCRIPTION,
    "Text=Sign out",
    "FrameName=UniversalHeader",
    ACTION,
    "UserAction=Click",
    LAST);

/*web_dump_cache("paycheck","FileName=c:\\{VuserName}paycheck",
"Replace=yes", LAST);*/
    return 0;
}
```

40

Web (HTTP/HTML, Click and Script) Text and Image Verification

You can add checks to your Web Vuser scripts to determine whether or not the correct Web pages are returned by the server when you run the Vuser script.

This chapter includes:

- About Verification Under Load on page 589
- Adding a Text Check on page 592
- Understanding Text Check Functions on page 594
- Adding an Image Check on page 600
- Defining Additional Properties on page 603

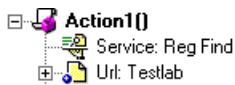
About Verification Under Load

VuGen enables you to add checks to your Web Vuser scripts. A Web check verifies the presence of a specific object on a Web page. The object can be a text string or an image.

Web checks enable you to determine whether or not your Web site is functioning correctly while it is being accessed by many Vusers—that is, does the server return the correct Web pages? This is particularly important while your site is under the load of many users, when the server is more likely to return incorrect pages.

For example, assume that your Web site displays information on the temperatures in major cities around the world. You use VuGen to create a Vuser script that accesses your Web site.

The Vuser accesses the site and executes a text check on this Web page. For example, if the word **Temperature** appears on the page, the check passes. If **Temperature** does not appear because, for example, the correct page was not returned by the server, the check fails. Note that the text check step appears before the URL step. This is because VuGen registers, or prepares in advance, the search information relevant for the next step. When you run the Vuser script, VuGen conducts the check on the Web page that follows.






Although the server may display the correct page when you record the script and when a single Vuser executes the script, it is possible that the correct page will not be returned when the server is under the load of many Vusers. The server may be overloaded and may therefore return meaningless or incorrect HTML code. Alternatively, in some instances when a server is overloaded, the server may return a **500 Server Error** page. In both of these cases, you can insert a check to determine whether or not the correct page is returned by the server.

Note: Web checks increase the work of a Vuser, and therefore you may need to run fewer Vusers per load generator. You should use Web checks only where experience has shown that the server sometimes returns an incorrect page.

You can define Web checks during or after recording a Vuser script.

VuGen uses several different Web check icons, each one representing a different check type:

Web Check Icon	Description
Text 	A text check, searching for a specific string in the next action function (web_reg_find) or in the entire business process (web_global_verification) step.
Text 	A text check, searching for a specific string in the downloaded HTML page using the web_find step. For more information, see "Understanding Text Check Functions" on page 594.
Image 	An image check, searching for a specific image on a Web page. For more information, see "Understanding Text Check Functions" on page 594.

This chapter describes how to use VuGen to add Web checks in the tree view. For information about adding checks to the script in the text-based script view, see the *Online Function Reference* (**Help > Function Reference**).

Adding a Text Check

VuGen allows you to add a check that searches for a text string on a Web page. You can add the text check either during or after recording.

When you create a text check, you define the name of the check, the scope of the check, the text you want to check for, and the search conditions.

To add a text check during recording:

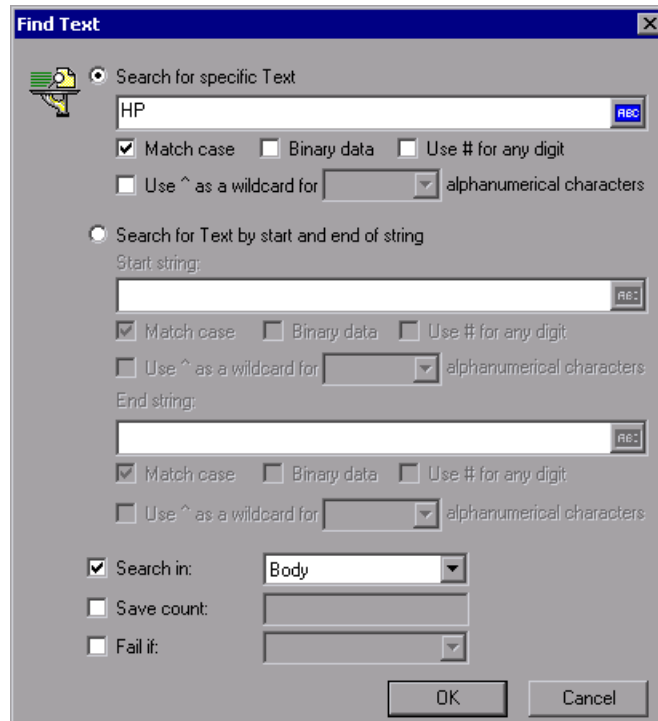
- 1 In the application or Web browser window, select the desired text.
- 2 Click the **Insert Text check** button on the recording toolbar. VuGen adds a `web_reg_find` function to the script.



To add a text check after recording:

- 1 Go to the snapshot of the step whose text you want to check.
- 2 In the snapshot, select the text you want to verify.
- 3 Select **Add a Text Check (web_reg_find)** from the right-click menu. The Find Text properties dialog box opens.

Note: For certain protocols, VuGen issues a message indicating that you should add text checks from the Server Response tab—not from the snapshot. Click the **Server Response** tab and select the **HTML Document** tab. Expand the body node and then continue as described below.



The following attributes are available for **web_reg_find**:

- **Search for specific Text.** The text string to search for. This attribute must be a non-empty, null-terminated character string.
- **Search for Text by start and end of string**
 - **Start string.** The prefix of the text string for which you are searching.
 - **End string.** The suffix of the text string for which you are searching.

After specifying a search method—specific text or a beginning or end string, you can specify the search options:

- ▶ To perform a case -sensitive search, select **Match case**. To indicate binary data, select **Binary data**. To indicate any digit as a match, use a hash (#) in the text string and select **Use # for any digit**.
- ▶ **Use ^ as a wildcard for all/lowercase/uppercase alphanumerical characters**. Allows a wildcard search for alphanumerical characters—either all, uppercase, or lowercase characters. You specify a wildcard with the ^ character.
- ▶ **Search in**. Where to search for the text. The available values are **Headers**, **Body**, or **All**. The default is **Body**.
- ▶ **Save count**. The number of matches that were found. To use this attribute, select **Save count** and specify a parameter name in which to store the number of matches. The variable will be a null-terminated ASCII value.
- ▶ **Fail if**. The handling method when the string is not found. The available values are **Found** and **Not Found**. **Found** indicates that a failure occurs when the text is found (for example, "Error"). **Not Found** indicates that a failure occurs when the text is not found.

To view or modify the properties of the text check after it has been created, click the **Tree View** tab and double-click on the **Services: Reg Find** step. In the Find Text dialog box, you can view or modify all of the step's attributes.

Understanding Text Check Functions

When you add a text check, VuGen adds a **web_reg_find** function to your script. This function registers a search for a text string on an HTML page. Registration means that it does not execute the search immediately—it performs the check only after executing the next Action function, such as **web_url**. Note that if you are working with a concurrent functions group, the **web_reg_find** function is only executed at the end of the grouping.

In the following example, **web_reg_find** function searches for the text string "Welcome". If the string is not found, the next action function fails and the script execution stops.

```
web_reg_find("Text=Welcome", "Fail=Found", LAST);  
web_url("Step", "URL=...", LAST);
```

In addition to the **web_reg_find** function, you can use other functions to search for text within an HTML page:

Several additional functions can be used for searching for text:

- **web_find**
- **web_global_verification**

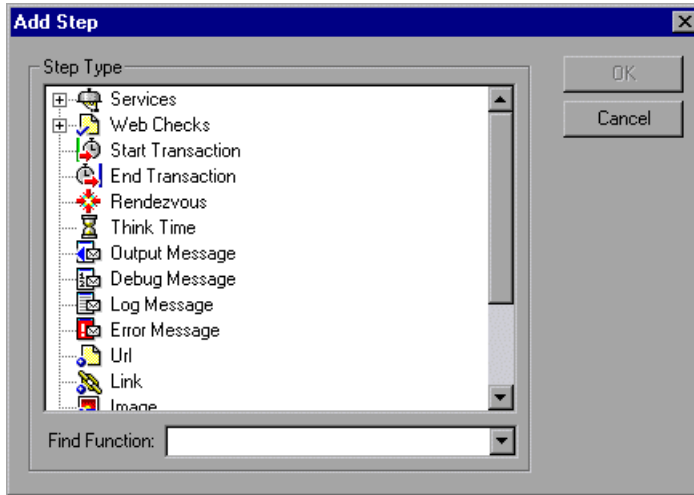
The **web_find** function, primarily used for backward compatibility, differs from the **web_reg_find** function in that **web_find** is limited to an HTML-based script (see **Recording Options > Recording** tab). It also has less attributes such as **instance**, allowing you to determine the number of times the text appeared. When performing a standard text search, **web_reg_find** is the preferred function.

The **web_global_verification** function allows you to search the data of an entire business process. In contrast to **web_reg_find**, which only applies to the next Action function, this function applies to **all** subsequent Action functions such as **web_url**. By default, the scope of the search is **NORESOURCE**, searching only the HTML body, excluding headers and resources.

The **web_global_verification** function is ideal for detecting application level errors that are not included the HTTP status codes. This function is not limited to an HTML-Based script (see **Recording Options > Recording** tab).

To add additional functions to your script:

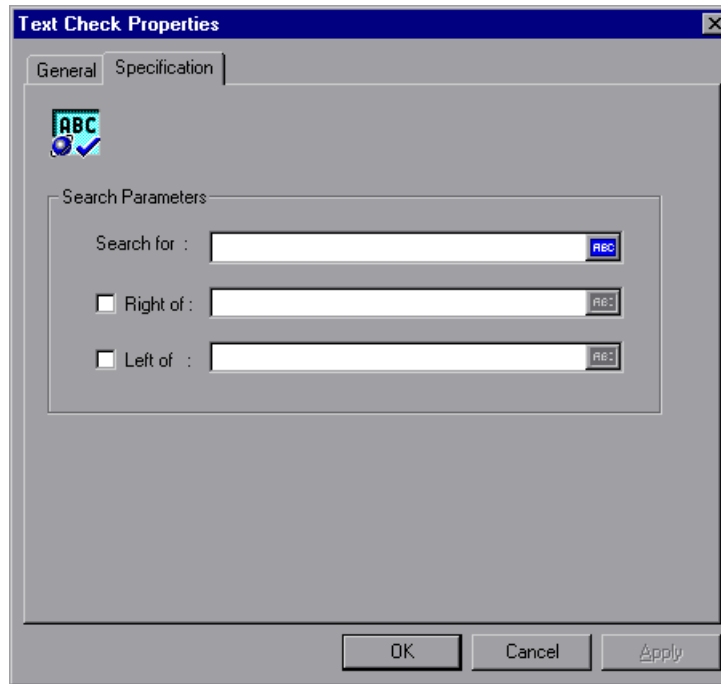
- 1 In the VuGen main window, click at the point where you want to add the text check. Select **Insert > New Step**.



- 2 For the **web_find** functions, expand the **Web Checks** node and select **Text Check**. For the **web_global_verification** function, expand the **Services** node and select the function name. The Properties dialog box opens.
- 3 Set the properties for these functions (see description below).
- 4 Click **OK**. VuGen inserts a new function into the script.

Setting web_find Properties

You can set the following properties for the `web_find` function:

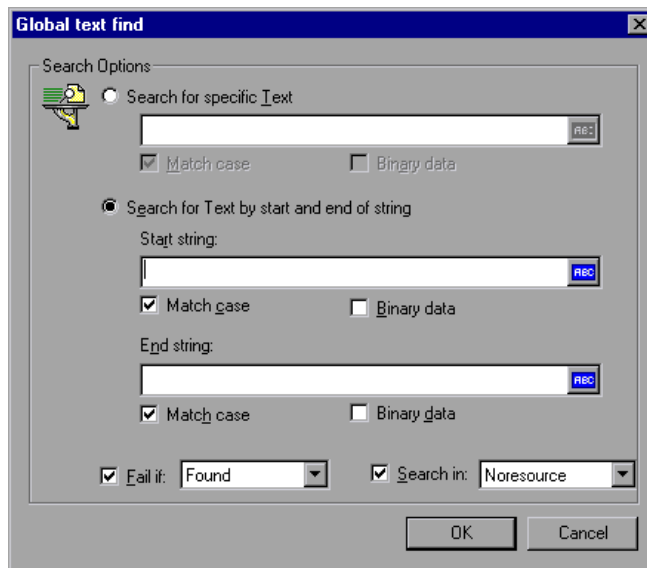


- **Search for.** The string you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.
- **Right of / Left of.** The position of the search string relative to adjacent text. Type the text in the appropriate field. For example, to verify that the string `support@hp.com` appears to the right of the word `e-mail:`, select **Right of** and then type `e-mail:` in the **Right of** box.
- **Step Name.** The name of the text check. Click the **General** tab and type a name for the text check in the box. Use a name that you can recognize and identify later on.

Note: A Vuser conducts Web checks during script execution only if checks are enabled, and if the script runs in HTML mode. To enable checks, select the **Enable image and text check** option in the **Preferences** tab in the Run-Time Settings dialog box. For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

Setting web_global_verification Properties

You can set the following properties for the **web_find** function:



- ▶ **Search for specific text.** The string whose presence you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.
- ▶ **Search for Text by Start and End of String.** The boundaries, also known as **Start** and **End** strings that surround the text. Select the appropriate options to indicate if you want to **Match case** or if you are searching for **binary data**.

- **Fail if.** Fails the script if the condition is met. You can also indicate the failure condition: if the text is **Found** or **Not found**. Select the desired behavior in the **Fail if** box.

Text Flags

When specifying search text using a registered search, **web_reg_find**, you can add flags to control the scope of the search:

/IC to ignore the case.

/BIN to specify binary data.

/DIG to interpret the pound sign (#) as a wildcard for a single digit. The **DIG** flag does not match a literal pound sign.

/ALNUM<case> to interpret the caret sign (^) as a wildcard for a single US-ASCII alphanumeric character. There are three syntaxes: **ALNUMIC** to ignore case, **ALNUMLC** to match only lower case, and **ALNUMUC** to match only upper case. The **ALNUM** flag does not match a literal caret.

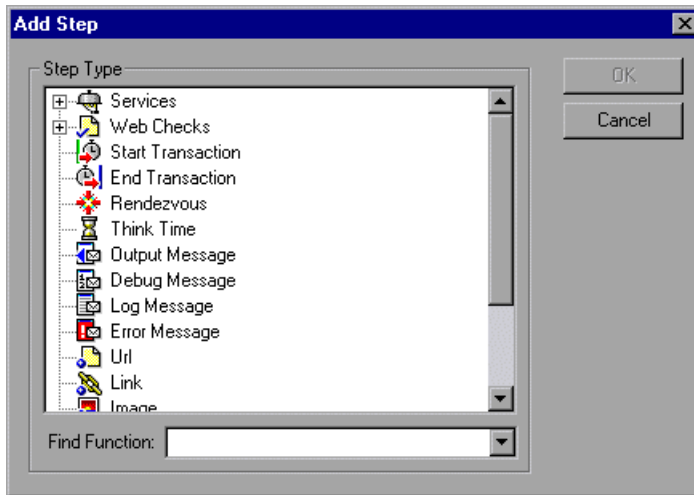
To use flags, you enter the attribute **TEXT**, followed by a forward slash and the flag name. For example, to search for a string ignoring the case, use "Text/IC=search_text".

Adding an Image Check

VuGen allows you to add a user-defined check that searches for an image on a Web page. The image can be identified by the ALT attribute, the SRC attribute, or both.

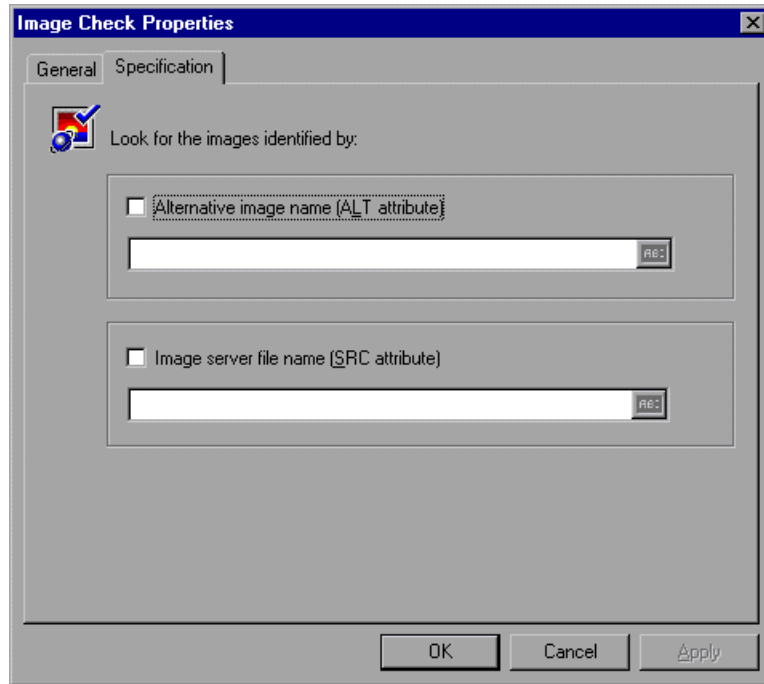
To add an image check:

- 1 In the VuGen main window, right-click the step corresponding to the Web page on which you want to perform a check. Select **Insert After** from the pop-up menu. The Add Step dialog box opens.



- 2 Expand **Web Checks** in the **Step Type** tree.

- 3 Select **Image Check**, and click **OK**. The Image Check Properties dialog box opens. Make sure that the **Specification** tab is visible.

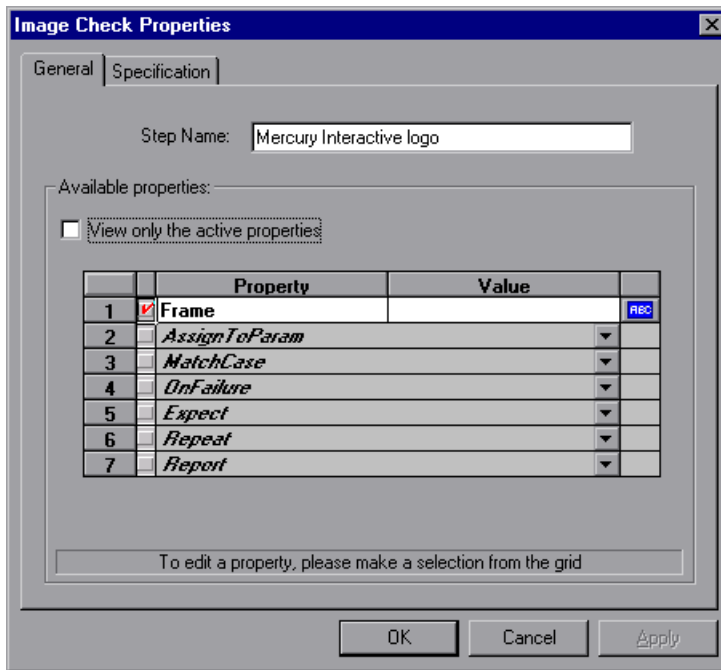


- 4 Select a method to identify the image:
 - **Alternative image name (ALT attribute)**. Identifies the image using its **ALT** attribute. Type the ALT attribute text in the text box. When you run the script, the Vuser searches for an image that has the specified ALT attribute.
 - **Image server file name (SRC attribute)**. Identifies the image using the SRC attribute. Type the SRC attribute text into the text box. When you run the script, the Vuser searches for an image that has the specified SRC attribute.

An ABC icon indicates that the ALT or SRC attribute has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.

Note: If you select both the ALT attribute and SRC attribute check boxes, the Vuser searches for an image that has both the specified ALT attribute and the specified SRC attribute.

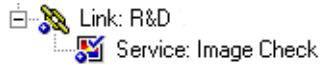
- To name the image check, click the **General** tab. In the **Step Name** box, type a name for the image check. Use a name that you can recognize later on.



- The properties table displays additional properties that define the check. Clear the **View only the active properties** check box to view active and non- active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column.

For details on assigning property values, see "Defining Additional Properties" on page 603.

- 7 Click **OK** to accept the settings. An icon representing the new image check is added to the associated step in the Vuser script.



Defining Additional Properties

You can specify additional properties for each Web check that you insert into a Vuser script. You set additional options in the properties table on the **General** tab of the check properties dialog boxes. The following is only relevant for **web_find** and **web_image_check** functions—not **web_reg_find**.

To set additional properties:

- 1 Right-click the Web check whose properties you want to edit, and select **Properties** from the pop-up menu. The appropriate check properties dialog box opens. Make sure that the **General** tab is visible.
- 2 Clear the **View only the active properties** check box to view all the available properties.
- 3 To enable a property, click the cell to the left of the property name. A red check mark appears beside the property.
- 4 Assign the property a value in the **Value** column:
 - **Frame.** Type the name of the frame where the check object is located.
 - **AssignToParam.** Select **YES** to enable assigning to a parameter. Select **NO** to disable this capability. The default value is **NO**.
 - **MatchCase.** Select **YES** to conduct a case-sensitive search. Select **NO** to conduct a non case-sensitive search. The default value is **NO**.

- ▶ **OnFailure.** Select **Abort** to abort the entire Vuser script if the check fails. VuGen aborts the Vuser script regardless of the error-handling method that has been set in the run-time settings. Select **Continue** to have the error-handling method defined in the run-time settings determine whether or not the script is aborted if the check fails.

The default value is **Continue**. For details on defining the error handling method, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

- ▶ **Expect.** Select **NotFound** to indicate that the check is successful if the Vuser does not find the specified check object. Select **Found** to indicate that the check is successful if the Vuser finds the specified check object. The default value is Found.
- ▶ **Repeat.** Select **YES** to search for multiple occurrences of the specified check object. Select **NO** to end the check as soon as one occurrence of the specified check object is found. The Vuser script continues with the next step. This option is useful when searching through a Web page that may have multiple occurrences of the check object. The default value is YES.
- ▶ **Report.** Select **Always** to always view a detailed description of the check results in the Execution Log. Select **Failure** to view detailed check results only when the check fails. Select **Success** to view detailed check results only when the check succeeds. The default value is **Always**.

An ABC icon indicates that the property value has not been assigned a parameter. Click the icon to assign a parameter. For more information, see "Creating Parameters" in *Volume I-Using VuGen*.

41

Modifying Web and Wireless Vuser Scripts

After recording a Web or Wireless Vuser script, you use VuGen to modify the recorded script. You can add new steps, and edit or delete existing steps.

This chapter includes:

- About Modifying Web and Wireless Vuser Scripts on page 606
- Adding a Step to a Vuser Script on page 607
- Deleting Steps from a Vuser Script on page 608
- Modifying Action Steps on page 609
- Modifying Control Steps on page 626
- Modifying Service Steps on page 629
- Modifying Web Checks (Web only) on page 630

About Modifying Web and Wireless Vuser Scripts

After recording a session, you can modify the recorded script in VuGen by editing a step's properties or adding and deleting steps.

You can do the modifications either in the icon-based tree view or in the text-based script view. For details on the two viewing modes, see Chapter 37, "Web (HTTP/HTML, Click and Script) Protocols."

This chapter describes how to use VuGen to modify the script in the tree view. For information about modifying the script in the text-based script view, see the *Online Function Reference* (**Help > Function Reference**).

Adding Binary Data

To include binary coded data in the body of an HTTP request, use the following format:

```
\x[char1][char2]
```

This represents the hexadecimal value that is represented by [char1][char2].

For example, \x24 is $16*2+4=36$, is a \$ sign, and \x2B is a + sign.

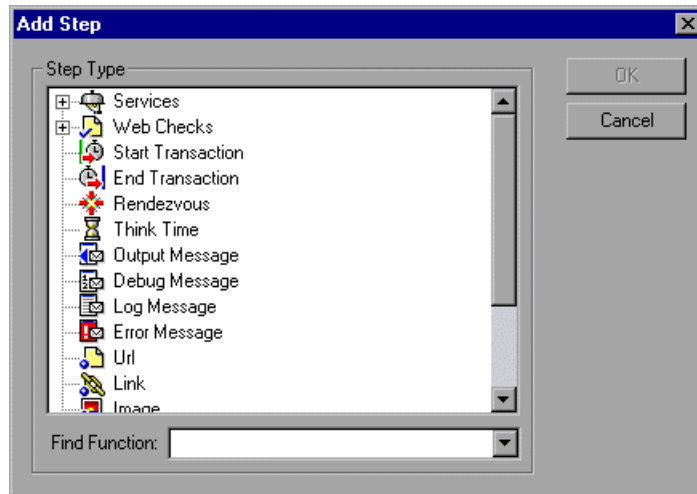
Do not use single-character hexadecimal sequences. For example, \x2 is not a valid sequence but \x02 is.

Adding a Step to a Vuser Script

In addition to the steps that VuGen records during the recording session, you can add steps to a recorded script.

To add a step to a Vuser script:

- 1 In the tree view of the script, select the step before or after which you want to add the new step.
- 2 Select **Insert > New Step** to insert a step after the selected step, or select **Insert After** or **Insert Before** from the right-click menu. The Add Step dialog box opens.



- 3 Select the type of step you want to add from the **Step Type** tree or from the **Find Function** list.
- 4 Click **OK**. An additional dialog box opens, prompting for information about the step to add. This dialog box varies, depending on the type of step that you are adding.

For details on using these dialog boxes, see the appropriate section, as listed below:

To add this...	See...
Vuser API function	"Enhancing Vuser Scripts" in <i>Volume I-Using VuGen</i>
Service step	"Modifying Service Steps" on page 629
Web Check	"Modifying Web Checks (Web only)" on page 630
Transaction	"Modifying a Transaction" on page 626
Rendezvous point	"Modifying a Rendezvous Point" on page 627
Think time step	"Modifying Think Time" on page 628
URL step	"Modifying a URL Step" on page 609
Link step	"Modifying a Hypertext Link Step (Web only)" on page 612
Image step	"Modifying an Image Step (Web only)" on page 613
Submit form step	"Modifying a Submit Form Step (Web only)" on page 615
Submit data step	"Modifying a Submit Data Step" on page 619
Custom request step	"Modifying a Custom Request Step" on page 623
User-defined step	"Enhancing Vuser Scripts" in <i>Volume I-Using VuGen</i>

Deleting Steps from a Vuser Script

After recording a session, you can use VuGen to delete any step from the Vuser script.

To delete a step from a Vuser script:

- 1** In the tree view of the Vuser script, right-click the step you want to delete, and select **Delete** from the pop-up menu.
- 2** Click **OK** to confirm that you want to delete the step.

The step is deleted from the script.

Modifying Action Steps

An action step represents a user action during recording, that is, a jump to a new URL or a change in the Web context.

Action steps, represented in the tree view of the Vuser script by Action icons, are added to your script automatically during recording. After recording, you can modify the recorded action steps.

This section includes:

- ▶ Modifying a URL Step
- ▶ Modifying a Hypertext Link Step (Web only)
- ▶ Modifying an Image Step (Web only)
- ▶ Modifying a Submit Form Step (Web only)
- ▶ Modifying a Submit Data Step
- ▶ Modifying a Custom Request Step

Modifying a URL Step

A URL step is added to the Vuser script when you type in a URL or use a bookmark to access a specific Web page.

The properties that you can modify are the name of the step, the address of the URL, target frame, and record mode.

By default, VuGen runs the URL step, based on the mode in which it was recorded: **HTML**, or **HTTP** (without resources). For information on the recording modes, see "Understanding the Recording Levels" on page 382 in *Volume I-Using VuGen*.

Setting the Replay Mode

In the URL step's Properties dialog box, you can modify the mode settings to instruct Vusers to execute the script in a mode other than the recorded mode. To customize the replay mode, select the **Record mode** check box. The available replay modes are:

- ▶ **HTML.** Automatically download all resources and images and store the required HTTP information for the steps that follow. This is ideal for script with Web links.
- ▶ **HTTP.** Do not download any resources for this step during replay. Download only resources that are explicitly represented by functions.

You can also indicate that a certain step should not be counted as a resource. For example, if you have a step that represents a specific image that you want to skip, you can instruct VuGen to exclude that resource type. For more information, see the Chapter 20, "Setting Recording Options for Web Vusers" in *Volume I-Using VuGen*.

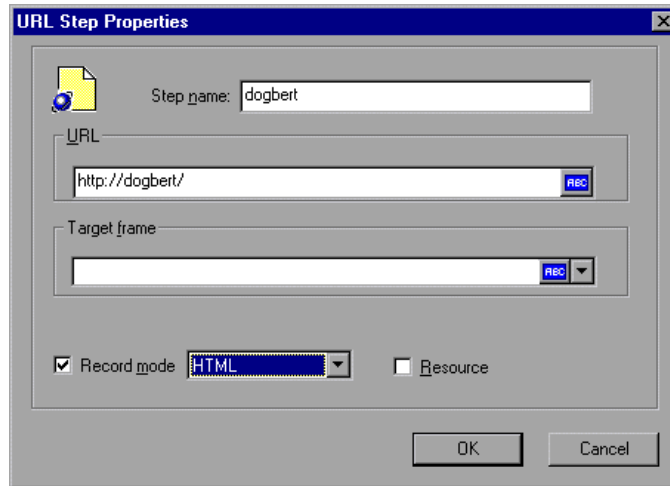
To modify the properties of a URL step:



- 1 In the tree view of the Vuser script, select the URL step you want to edit. URL steps are shown using the URL icon.



- 2 Click the **Properties** button on the VuGen toolbar. The URL Step Properties dialog box opens.



- 3 To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.
- 4 In the **URL** box, type the Web address (URL) of the Web page that is accessed by the URL step. An **ABC** icon indicates that the URL has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.
- 5 In the **Target frame** list, select one of the following values:
 - **SELF**. Replaces the last (changed) frame.
 - **PARENT**. Replaces the parent of the last (changed) frame.
 - **TOP**. Replaces the whole page.
 - **BLANK**. Opens a new window.
- 6 To customize the replay mode, select the **Record mode** check box. Select the desired mode: HTML or HTTP.
- 7 To exclude an item from being downloaded as a resource, clear the **Resource** check box.
- 8 Click **OK** to close the URL Step Properties dialog box.

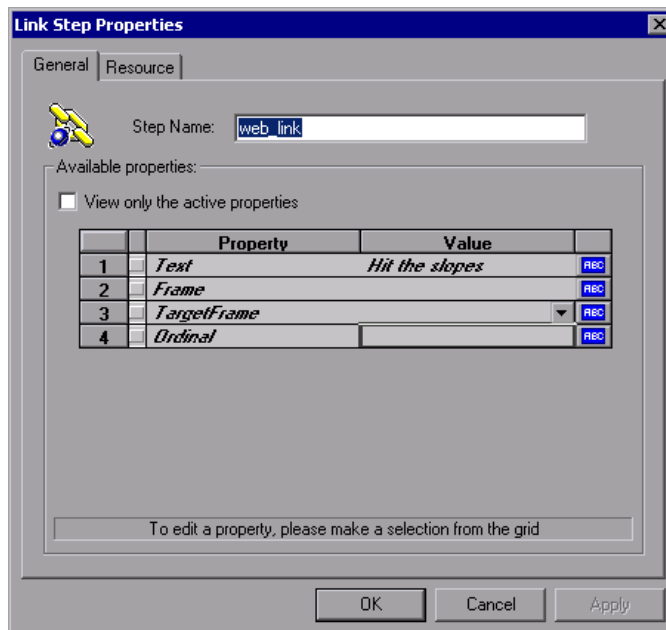
Modifying a Hypertext Link Step (Web only)

A hypertext link step is added to the Web Vuser script when you click a hypertext link. This step is only recorded when you select the option to record in **HTML based script** mode. See Chapter 20, "Setting Recording Options for Web Vusers." in *Volume I-Using VuGen* for more information.

The properties that you can modify are the name of the step, how the hypertext link is identified, and where it is located.

To modify the properties of a hypertext link step:

- 1 In the tree view of the Vuser script, select the hypertext link step you want to edit. Hypertext link steps are shown using the **Hypertext Link** icon.
- 2 Select **Properties** from the right-click menu. The Link Step Properties dialog box opens.



- 3 To change the step name, type a new name in the **Step Name** box. The default name during recording is the text string of the hypertext link.
- 4 The properties table displays the properties that identify the link.

Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **Text.** The exact string of the hypertext link.
- **Frame.** The name of the frame where the link is located.
- **TargetFrame.** The target frame:
 - **TOP.** Replaces the whole page.
 - **BLANK.** Opens a new window.
 - **PARENT.** Replaces the parent of the last (changed) frame.
 - **SELF.** Replaces the last (changed) frame.
- **Ordinal.** a number that uniquely identifies the link when all the other property attributes are identical to one or more other links on the Web page. See the *Online Function Reference* for details.

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Click **OK** to close the Link Step Properties dialog box.

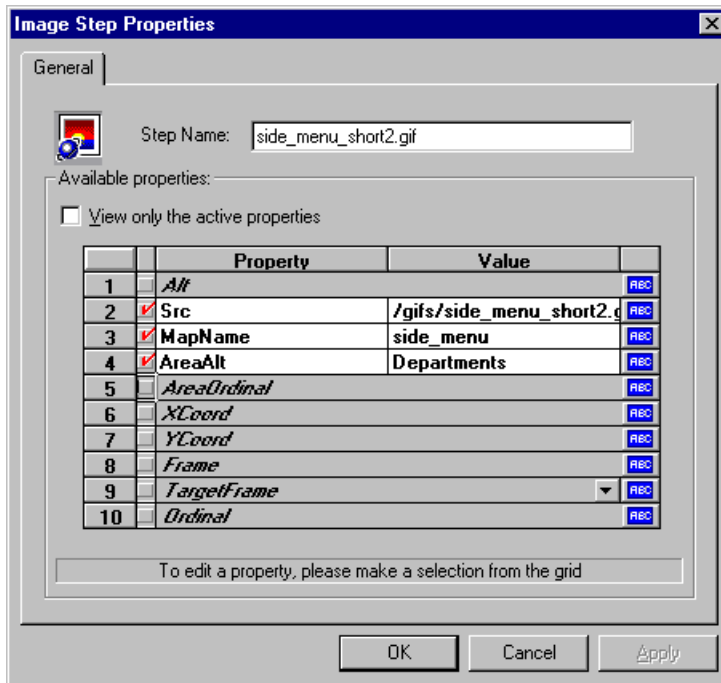
Modifying an Image Step (Web only)

An image step is added to the Vuser script when you click a hypergraphic link. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. See Chapter 20, "Setting Recording Options for Web Vusers." in *Volume I-Using VuGen* for more information.

The properties that you can modify are the name of the step, how the hypergraphic link is identified, and where it is located.

To modify the properties of an image step:

- 1 In the tree view of the Vuser script, select the image step you want to edit. Image steps are shown using the **Image** icon.
- 2 Select **Properties** from the right-click menu. The Image Step Properties dialog box opens.



- 3 To change the step name, type a new name in the **Step Name** box. The default name during recording is the image's ALT attribute. If the image does not have an ALT attribute, then the last part of the SRC attribute is used as the default name.
- 4 The properties table displays the properties that identify the link. Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:
 - **ALT.** The ALT attribute of the image.

- **SRC.** The SRC attribute of the image.
- **MapName.** The name of the map related to the image. Applies to client-side image maps only.
- **AreaAlt.** The ALT attribute of the area to click. Applies to client-side image maps only.
- **AreaOrdinal.** The serial number of the area to click. Applies to client-side image maps only.
- **Frame.** The name of the frame where the image is located.
- **TargetFrame.** The target frame:
 - **_TOP.** Replaces the whole page.
 - **_BLANK.** Opens a new window.
 - **_PARENT.** Replaces the parent of the last (changed) frame.
 - **_SELF.** Replaces the last (changed) frame.
- **Ordinal.** a number that uniquely identifies the image when all other property attributes are identical to one or more other images on the Web page. See the *Online Function Reference* for details.
- **XCoord, YCoord.** The coordinates of the mouse-click on the image.

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Click **OK** to close the Image Step Properties dialog box.

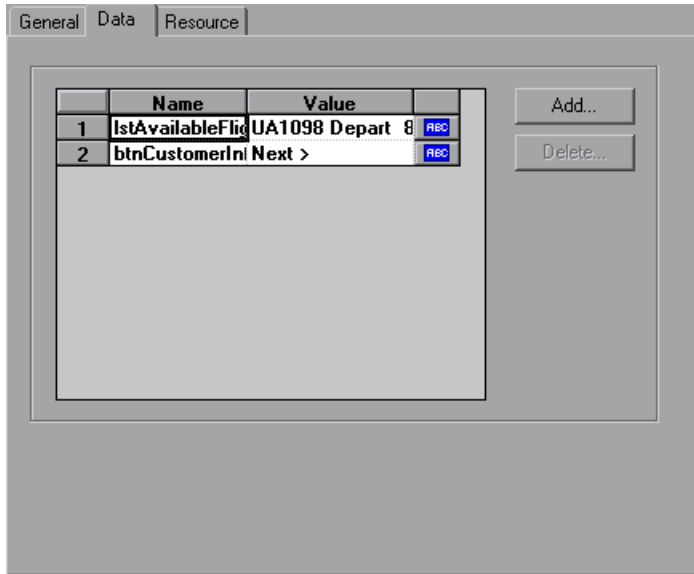
Modifying a Submit Form Step (Web only)

A submit form step is added to the Vuser script when you submit a form. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. See Chapter 20, "Setting Recording Options for Web Vusers." in *Volume I-Using VuGen* for more information.

The properties that you can modify are the name of the step, the form location, how the form submission is identified, the form data, and the resources for the step.

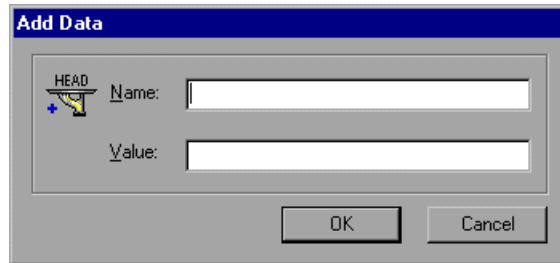
To modify the properties of a submit form step:

- 1** In the tree view of the Vuser script, select the submit form step you want to edit. Submit form steps are shown using the **Submit Form** icon.
- 2** Select **Properties** from the right-click menu. The Submit Form Step Properties dialog box opens. Click the **Data** tab.



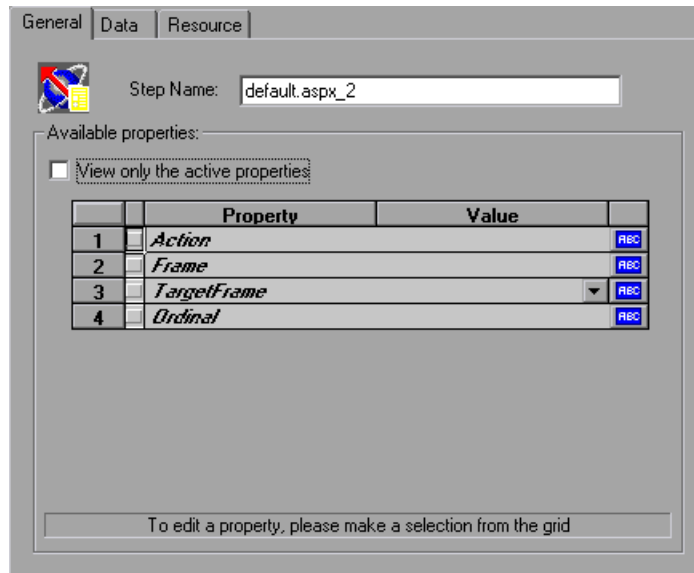
- The **Name** column lists all the data arguments on the form.
 - The **Value** column displays the corresponding value input for a data argument.
 - The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in "Creating Parameters" in *Volume I-Using VuGen*, the ABC icon changes to a table icon.
- 3** To edit a data argument, double-click on it to activate the cursor within the cell and type the new value in the editable box.

- 4 To add a new data argument to the form submission, click **Add**. The Add Data dialog box opens.



The 'Add Data' dialog box has a blue title bar. Inside, there is a 'HEAD' icon with a plus sign and a dropdown arrow. Below it are two text input fields: 'Name:' and 'Value:'. At the bottom are 'OK' and 'Cancel' buttons.

- 5 Type a **Name** and **Value** for the data argument, and click **OK**.
- 6 To delete an argument, select it and click **Delete**.
- 7 To change the name of the submit form step, click the **General** tab.



The 'General' tab shows a 'Step Name' field with the value 'default.aspx_2'. Below it is a section for 'Available properties' with a checkbox 'View only the active properties'. A table lists properties with their values and a 'ABC' button for each.

	Property	Value	
1	Action		ABC
2	Frame		ABC
3	TargetFrame		ABC
4	Ordinal		ABC

At the bottom, a text box contains the instruction: 'To edit a property, please make a selection from the grid'.

- 8 To change the step name, type a new name in the **Step Name** box. The default name during recording is the name of the executable program used to process the form.

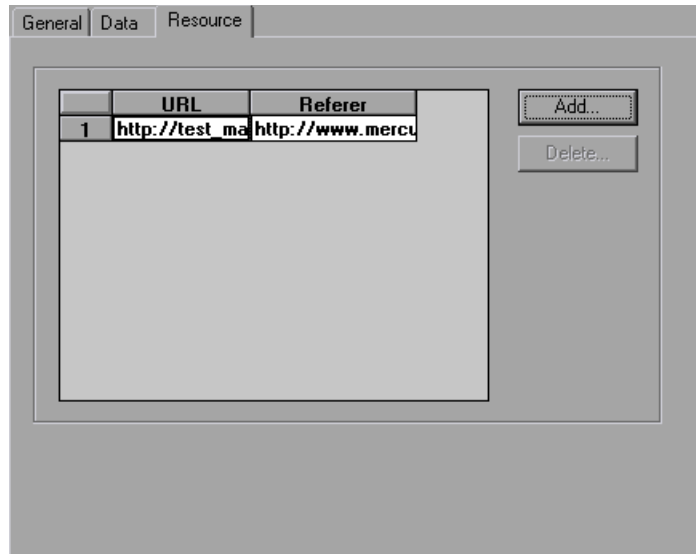
- 9 The properties table displays the properties that identify the form submission.

Clear the **View only the active properties** option to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **Action.** The address to be used to carry out the action of the form.
- **Frame.** The name of the frame where the form submission is located.
- **TargetFrame.** The target frame:
 - **_TOP.** Replaces the whole page.
 - **_BLANK.** Opens a new window.
 - **_PARENT.** Replaces the parent of the last (changed) frame.
 - **_SELF.** Replaces the last (changed) frame.
- **Ordinal.** a number that uniquely identifies the form when all other property attributes are identical to one or more other forms on the same Web page. See the *Online Function Reference* for details (**Help > Function Reference**).

An ABC icon indicates that the submit form step property value has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.

- 10** To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.



- 11** Click **OK** to close the Submit Form Step Properties dialog box.

Modifying a Submit Data Step

A submit data step represents the submission of data to your Web site for processing. This is different from a Submit Form step because you do not need to have a form context to execute this request.

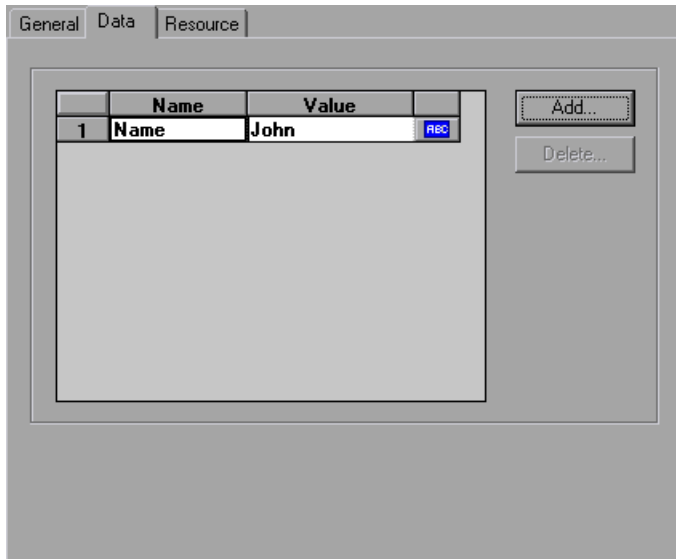
The properties that you can modify are the name of the step, the method, the action, the target frame, and the data items on the form.

To modify the properties of a submit data step:



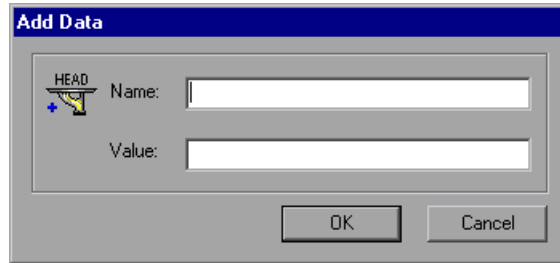
- 1** In the tree view of the Vuser script, select the submit data step you want to edit. Submit data steps are represented by the **Submit Data** icon.

- 2 Select **Properties** from the right-click menu. The Submit Data Step Properties dialog box opens. Click the **Data** tab.



- The **Name** column lists all the data arguments on the form. This includes all hidden fields.
 - The **Value** column displays the corresponding value input for a data argument.
 - The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in "Creating Parameters" in *Volume I-Using VuGen*, the **ABC** icon changes to a table icon.
- 3 To edit a data argument, double-click on it to activate the cursor within the cell. Then type the new value.

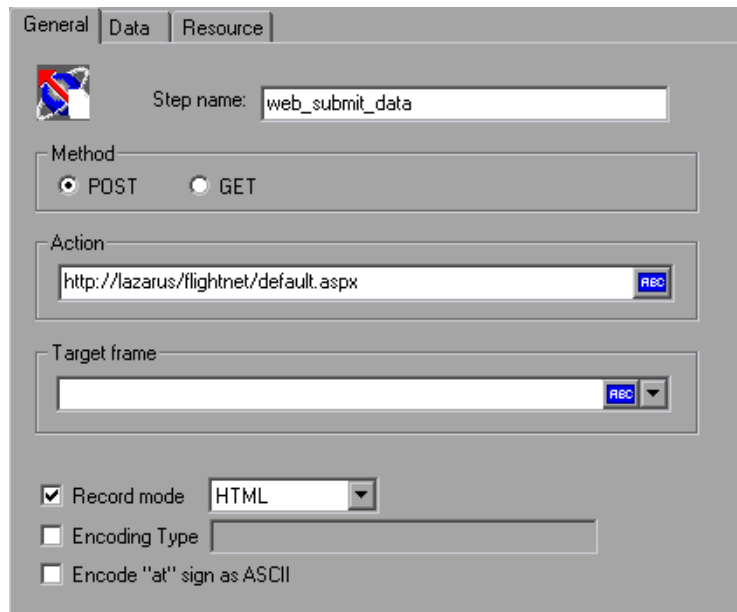
4 To add new data, click **Add**. The Add Data dialog box opens.



5 Type a **Name** and **Value** for the data argument, and click **OK**.

6 To delete an argument, select it and click **Delete**.

7 To change the name of the submit data step, click the **General** tab.

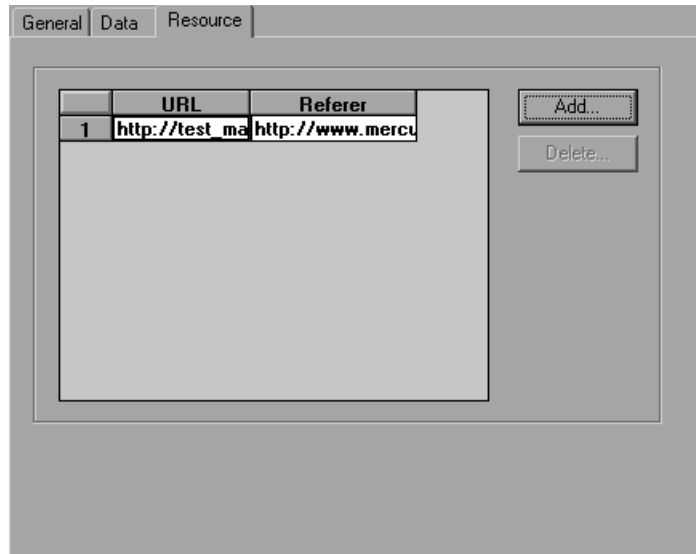


8 To change the step name, type a new name in the **Step name** box.

9 Under **Method**, click **POST** or **GET**. The default method is **POST**.

- 10** In the **Action** box, type the address to be used to carry out the action of the data submission. An ABC icon indicates that the action has not been assigned a parameter. For details on assigning parameters, see "Creating Parameters" in *Volume I-Using VuGen*.
- 11** Select a **Target frame** from the list:
 - **TOP**. Replaces the whole page.
 - **BLANK**. Opens a new window.
 - **PARENT**. Replaces the parent of the last (changed) frame.
 - **SELF**. Replaces the last (changed) frame.
- 12** To customize the replay mode, select the **Record mode** option. Select the desired mode: HTML, or HTTP. For more information, see "Setting the Replay Mode" on page 610.
- 13** To specify an encoding type, such as **multipart/www-urlencoded**, select the **Encoding type** check box and specify the encoding method.
- 14** To encode the "@" in the URL, select **Encode "at" sign as ASCII**.
- 15** Click **OK** to close the Submit Data Step Properties dialog box.

- 16** To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.



Modifying a Custom Request Step

A custom request represents a custom HTTP request for a URL, with any method supported by HTTP. A custom request step is contextless.

The properties that you can modify are the name of the step, method, URL, target frame, and body.

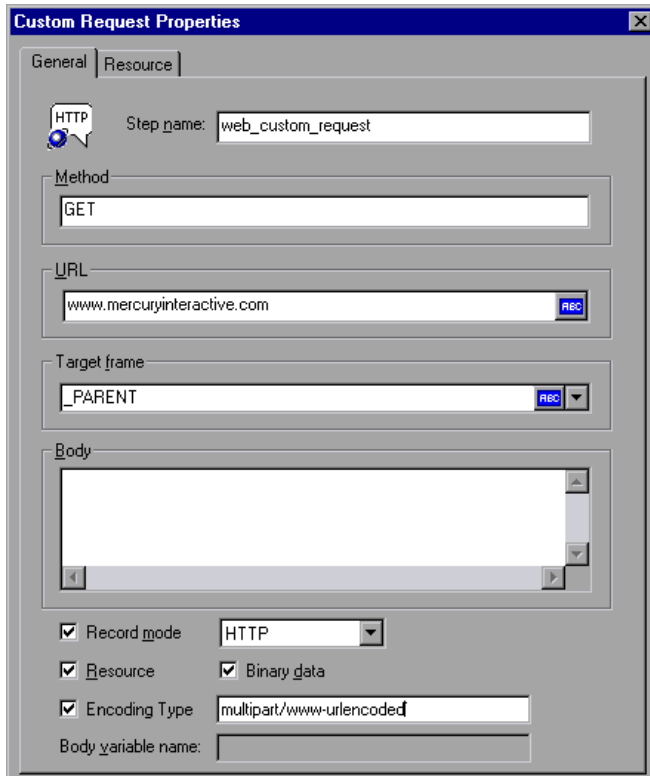
VuGen has a feature that lets you convert a custom request body string to C format. For example, if you copy an XML tree or a large amount of data into the Body area of the custom request, you can convert the strings to C format in order that it may be incorporated into the current function. VuGen inserts the necessary escape sequence characters and removes the line breaks in the string.

To modify the properties of a custom request step:

- 1** In the tree view of the Vuser script, select the custom request step you want to edit. Custom request steps are shown using the **Custom Request** icon.



- 2 Select **Properties** from the right-click menu. The Custom Request Properties dialog box opens.



- 3 To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.
- 4 In the **Method** box, type any method supported by HTTP. For example, GET, POST or HEAD.
- 5 In the **URL** box, type the URL being requested.
- 6 Select a **Target frame** from the list:
 - **TOP.** Replaces the whole page.
 - **BLANK.** Opens a new window.
 - **PARENT.** Replaces the parent of the last (changed) frame.
 - **SELF.** Replaces the last (changed) frame.

- 7** In the **Body** attribute box, type the body of the request or paste in the desired text. If you select the **Binary data** check box, the text is treated as binary and not as ASCII. For details on using binary data, see the *Online Function Reference* (**Help > Function Reference**).

VuGen replaces a Body attribute whose length exceeds 100K, with a variable in the **Body variable name** box. The variable is defined in the **lrw_custom_body.h** file located in the **include** folder.

- 8** For strings that you pasted into the **Body** box, select the text and select **Convert to C format** from the right-click menu.
- 9** To customize the replay mode, select the **Record mode** option. Select the desired mode: HTML or HTTP. For more information, see "Setting the Replay Mode" on page 610.
- 10** To exclude an item from being downloaded as a resource, clear the **Resource** option.
- 11** To specify an encoding type, such as **multipart/www-urlencoded**, select **Encoding type** and specify the encoding method.
- 12** Click **OK** to close the Custom Request Properties dialog box.

Modifying Control Steps

A control step represents a non-action step used during load testing. Control steps include transactions, rendezvous points, and think time.

You add control steps, represented in the tree view of the Vuser script by Control icons, to your script during and after recording.

This section includes:

- ▶ Modifying a Transaction
- ▶ Modifying a Rendezvous Point
- ▶ Modifying Think Time

Modifying a Transaction

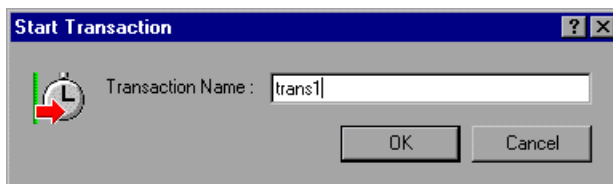
A transaction is a task or set of actions whose server response time you want to measure.

The properties that you can modify are the name of the transaction (start transaction and end transaction) and its status (end transaction only).

To modify a start transaction control step:



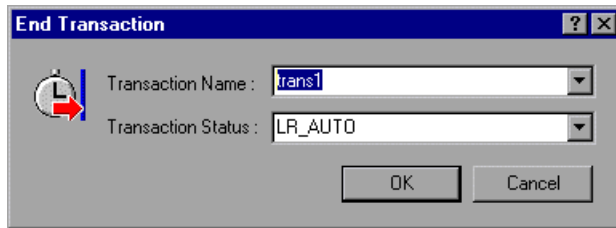
- 1** In the tree view of the Vuser script, select the start transaction control step you want to edit. Start transaction control steps are shown using the **Start Transaction** icon.
- 2** Select **Properties** from the right-click menu. The Start Transaction dialog box opens.



- 3** To change the transaction name, type a new name in the **Transaction Name** box, and click **OK**.

To modify an end transaction control step:

- 1 In the tree view of the Vuser script, select the end transaction control step you want to edit. End transaction control steps are shown using the **End Transaction** icon.
- 2 Select **Properties** from the right-click menu. The End Transaction dialog box opens.



- 3 Select the name of the transaction you want to end from the **Transaction Name** list.
- 4 Select a transaction status from the **Transaction Status** list:
 - **LR_PASS**. Returns a "succeed" return code.
 - **LR_FAIL**. Returns a "fail" return code.
 - **LR_STOP**. Returns a "stop" return code.
 - **LR_AUTO**. Automatically returns the detected status.

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

- 5 Click **OK** to close the End Transaction dialog box.

Modifying a Rendezvous Point

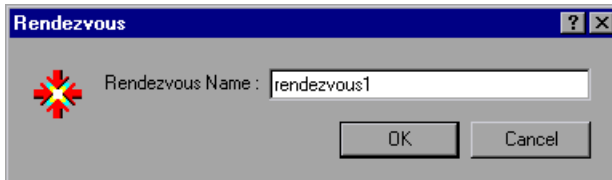
Rendezvous points enable you to synchronize Vusers to perform a task at exactly the same moment.

The property that you can modify is the name of the rendezvous point.

To modify a rendezvous point:

- 1 In the tree view of the Vuser script, select the rendezvous point you want to edit. Rendezvous points are shown using the **Rendezvous** icon.

- 2 Select **Properties** from the right-click menu. The Rendezvous dialog box opens.



- 3 To change the rendezvous name, type a new name in the **Rendezvous Name** box, and click **OK**.

Modifying Think Time

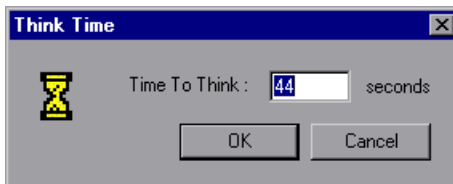
Think time emulates the time that a real user waits between actions. During recording, VuGen automatically adds think time to the Vuser script after each user action—if the time between that action and the subsequent action exceeds the predefined threshold.

The property that you can modify is the think time, in seconds.

To modify the think time:



- 1 In the tree view of the Vuser script, select the think time step you want to edit. Think time steps are shown using the **Think Time** icon.
- 2 Select **Properties** from the right-click menu. The Think Time dialog box opens.



- 3 Type a think time in the **Time To Think** box, and click **OK**.

Note: When you run a Web Vuser script, you can instruct the Vuser to replay think time as recorded or ignore the recorded think time. For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

Modifying Service Steps

A service step is a function that performs customization tasks such as setting proxies, submitting authorization information, and issuing customized headers. Service steps do not make any changes to the Web site context.

You add service steps to your script during and after recording.

To modify the properties of a service step:



- 1** In the tree view of the Vuser script, select the service step you want to edit. Service steps are shown using the Service icon.
- 2** Select **Properties** from the right-click menu. The appropriate service step properties dialog box opens. This dialog box varies, depending on the type of service step that you are modifying. A description of the service step is displayed in the title bar of the dialog box.

Note: Some service step functions have no arguments. In these cases, the Properties menu item is disabled.

- 3** Type or select the arguments required for the service step. See the *Online Function Reference* for details of each function (**Help > Function Reference**).
- 4** Click **OK** to close the service step properties dialog box.



Modifying Web Checks (Web only)

A Web check is a function that verifies the presence of a specific object on a Web page. The object can be a text string or an image.

You add Web checks to your script during and after recording.

To modify the properties of a Web check:

- 1 In the tree view of the Vuser script, select the Web check you want to edit. Web checks are shown using Web Check icons.

Icon	Description
	Image Check icon
	Text Check icon

- 2 Select **Properties** from the right-click menu. The appropriate Web check properties dialog box opens. This dialog box varies, depending on the type of check that you are modifying.
- 3 Type or select the properties required for the check. For details, see Chapter 40, "Web (HTTP/HTML, Click and Script) Text and Image Verification."
- 4 Click **OK** to close the check properties dialog box.

42

Web (HTTP/HTML) Correlation Rules

VuGen's correlation feature allows you to link Vuser functions by using the results of one statement as input for another.

This chapter includes:

- ▶ About Correlating Statements on page 631
- ▶ Understanding the Correlation Methods on page 633
- ▶ Using VuGen's Correlation Rules on page 634
- ▶ Setting Correlation Rules on page 639
- ▶ Testing Rules on page 642

About Correlating Statements

HTML pages often contain dynamic data, which is data that changes each time you access a site. For example, certain Web servers use links comprised of the current date and time.

When you record a Web Vuser script, dynamic data may be recorded into the script. Your script tries to present the recorded values to the Web server, but they are no longer valid. The Web server rejects them and issues an error. These errors are not always obvious, and you may only detect them by carefully examining Vuser log files.

If you encounter an error when running your Vuser, examine the script at the point where the error occurred. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

The dynamic data in an HTML page can be in the form of:

- ▶ a URL that changes each time you access the associated Web page
- ▶ form field (sometimes hidden) value(s).
- ▶ JavaScript cookies

Case 1

Suppose a Web page contains a hypertext link with text: "Buy me now!"
When you record a script with HTTP data, the URL is recorded by VuGen as:

"http://host//cgi-bin/purchase.cgi?date=170308&ID=1234"

Since the date "170308" and ID "1234" are created dynamically during recording, each new browser session recreates the date and ID. When you run the script, the link "Buy me now!" is no longer associated with the same URL that was recorded—but with a new one. The Web server is therefore unable to retrieve the URL.

Case 2

Consider a case where a user fills in his name and account ID into a form, and then submits the form.

When the form is submitted, a unique serial number is also sent to the server together with the user's data. Although this serial number is contained in a hidden field in the HTML code, it is recorded by VuGen into the script. Because the serial number changes with each browser session, Vusers were unable to successfully replay the recorded script.

You can use correlated statements to resolve the difficulties in both of the above cases. Replace the dynamic data in the recorded script with one or more parameters. When the script runs, it assigns values to each of the parameters.

Understanding the Correlation Methods

This chapter discusses automatic correlation using built-in or user-defined rules. To manually correlate statements, or to perform correlation for Wireless Vuser scripts, see "Performing Manual Correlation" on page 655.

When recording a browser session, you should first try recording in HTML mode. This mode decreases the need for correlation. For more information about the various recording modes, see Chapter 20, "Setting Recording Options for Web Vusers" in *Volume I-Using VuGen*.

You can instruct VuGen to correlate the statements in your script either during or after recording. The recording-time solutions described in this chapter automatically correlate the statements in your script during recording time. You can also use VuGen's snapshot correlation to correlate scripts after recording. For more information on correlating after recording, see Chapter 43, "Web (HTTP/HTML) -Correlation After Recording."

Using VuGen's Correlation Rules

VuGen's correlation engine allows you to automatically correlate dynamic data during your recording session using one of the following mechanisms:

- ▶ Built-in Correlation
- ▶ User-Defined Rule Correlation

For additional information, see "Adding Match Criteria" on page 638 and "Advanced Correlation Rules" on page 638.

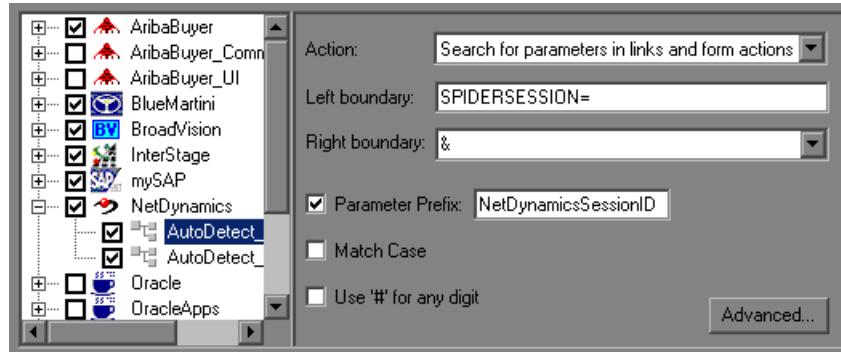
Built-in Correlation

The Built-in correlation detects and correlates dynamic data for supported application servers. Most servers have clear syntax rules, or contexts, that they use when creating links and referrals.

For example, BroadVision servers create session IDs that are always placed between the same delimiters: "BV_SessionID=" on the left, and "&" on the right.

```
BV_SessionID=@@@@1303778278.0969956817@@@@&
```

If you are recording a session with a supported application server, you can use one of the existing rules built into VuGen. An application server may have more than one rule. You can enable or disable a specific rule by selecting or clearing the check box adjacent to the rule. VuGen displays the rule definitions in the right pane.



If you are recording a session on an unsupported application server whose context is not known, and you cannot determine any correlation rules, you can use VuGen's snapshot comparison method. This method guides you through the correlation procedure after you finish recording. For more information, see Chapter 43, "Web (HTTP/HTML) -Correlation After Recording."

User-Defined Rule Correlation

If your application has unique rules and you are able to determine them clearly, you can define new rules using the Recording Options.

User-defined rule correlation requires you to define correlation rules before you record a session. You create the correlation rules in the Recording Options dialog box. The rules include information such as the boundaries of the dynamic data you want to correlate and other specifications about the match such as binary, case matching, and the instance number.

You instruct VuGen where to search for the criteria:

- All Body Text
- Link/Form Actions
- Cookie Headers
- Form Field Value
- Insert Cookie Function

Note that by default, the maximum size of a string that you can save for a rule is 4096 characters. If necessary, you can modify this value by increasing the value of the **MaxParamLen** attribute in the **CorrelationSettings.xml** file, located in the Windows Installation directory.

All Body Text

The **Search for Parameters in all of the Body Text** option instructs the recorder to search the entire body—not just links, form actions or cookies. It searches the text for a match using the borders that you specify.

Link/Form Actions

The **Search for parameters in links and form actions** method instructs VuGen to search within links and forms' actions for the text to parameterize. This method is for application servers where you know the context rules. You define a left boundary, a right boundary, an alternate right boundary, and an instance (occurrence) of the left boundary within the current link.

For example, suppose you want to replace any text between the second occurrence of the string "sessionId=" and "@" with a parameter. Specify **sessionId=** as a left boundary in the **Left Boundary** box, and **@** as a right boundary in the **Right Boundary** box. Since you are looking for the second occurrence, select **second** in the **Instance** box.

If the right boundary is not consistent, you can specify an alternate right boundary in the **Alternate right boundary** box. It uses this value when it cannot uniquely determine the specified right boundary.

For example, suppose the Web page contains links in the following formats:

```
"SessionID=122@page.htm"  
"Page.htm@SessionID=122&test.htm"
```

Specifying the right boundary alone is not sufficient, since it is not consistent—sometimes it is "@" and other times it is "&". In this case, you specify "&" as the alternate right boundary.

The left and right boundaries should uniquely identify the string. Do not include dynamic data in the boundaries. You can also specify **End of String** or **Newline Character** as a right boundary, available as options in the drop-down menu.

Note that for this option, the left and right boundaries must appear in the string that appears in the script—it is not sufficient for the boundaries to be returned by the server. This limitation does not apply to the other action types.

Cookie Headers

The **Search for Parameters from Cookie Header** method is similar to the previous rule, except that the value is extracted from cookie text (exactly as it appears in the recording log) instead of from a link or form action.

Form Field Value

The **Parameterize form field value** method instructs VuGen to save the named form field value to a parameter. It creates a parameter and places it in the script before the form's action step. For this option, you need to specify the field name.

Insert Cookie Function

The **Text to enter a web_reg_add_cookie function by** method inserts a **web_reg_add_cookie** function if it detects a certain string in the buffer. It only adds the function for those cookies with the specified prefix. For this option, you need to specify the search text and the cookie prefix.

Adding Match Criteria

In addition to the above rules, you can further define the type of match for your correlation by specifying the following items for the string:

- ▶ **Parameter Prefix.** Uses a prefix in all automatically generated parameters based on this rule. Prefixes prevent you from overwriting existing user parameters. In addition, prefixes allow you to recognize the parameter in your script. For example, in Siebel Web, one of the built-in rules searches for Siebel_row_id prefix.
- ▶ **Match Case.** Matches the case when looking for boundaries.
- ▶ **Use "#" for any digit.** Replaces all digits with a hash sign. The hash signs serve as wildcard, allowing you to find text strings with any digit. For example, if you enable this option and specify **HP###** as the left boundary, **HP193** and **HP284** will be valid matches.

Adding Comments

You can instruct VuGen to insert descriptive comments to the correlation steps within your script. To enable this option, select the **Add Comments to script** option.

Advanced Correlation Rules

VuGen lets you specify the following advanced correlation rules:

- ▶ **Always create new parameter.** Creates a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance. This option should be set if the Web server assigns a different value for each page. For example, NetDynamics servers may change the session ID from page to page to minimize fraud.
- ▶ **Replace with parameter only for exact matches.** Replace the recorded value with a parameter only when the text between the boundaries exactly matches the found value (from the first snapshot). If there are additional characters either before or after the string, it will not replace the parameter.

For example, in a form submission, VuGen recorded the characters 1234 between the boundaries aaa and bbb, `aaa1234bbb`. In subsequent submissions of this form, VuGen only replaces the recorded value with a parameter if it finds the characters 1234, `Name=1234`. If another value is entered, even if it contains the first string, for example, `Name=12345`, VuGen will not replace the value with a parameter. Instead, it will use the value 12345.

- ▶ **Reverse Search.** Searches for the left boundary from the end of the string backwards.
- ▶ **Left boundary Instance.** The number of occurrence of the left boundary within the string (not the body) for it to be considered a match.
- ▶ **Offset.** The offset of a sub-string of the found value to save to the parameter. The default is the beginning of the matched string. Note that you must specify a non-negative value.
- ▶ **Length.** The length from its offset of a sub-string of the matched string to save to the parameter. If you disable this option, the default saves the string from the specified offset until the end of the match.
- ▶ **Alternate Right Boundary.** An alternative criteria for the right boundary if the previously specified boundary is not found. You can specify text, **End of String**, or **Newline Character**.

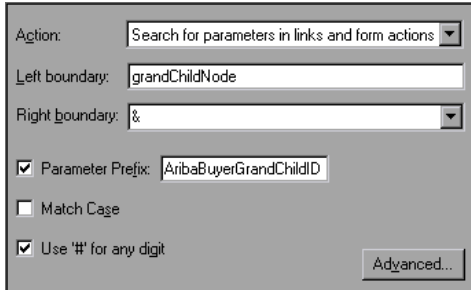
Setting Correlation Rules

You can add, modify, or remove rules using the Correlation Recording options. Note that you can also edit rules that were created automatically for application server environments.

In addition to creating rules using the recording options before recording, you can create rules after recording. After running your script, you scan it for correlations (CTRL+F8). You select one of the correlation results, and create a rule based on its properties. For more information, see "Performing a Scan for Correlations" on page 651.

To define correlation rules:

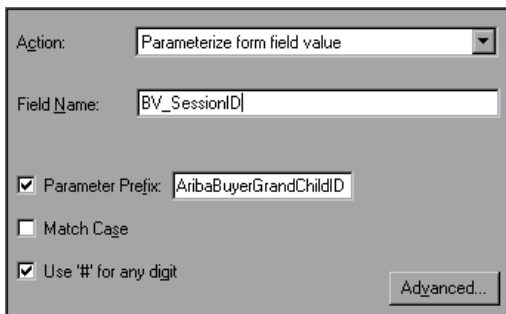
- 1 Click on an existing rule or click **New Rule** in the left pane. The Correlation Rules are displayed in the right pane.



The screenshot shows a dialog box with the following fields and options:

- Action: Search for parameters in links and form actions (dropdown)
- Left boundary: grandChildNode (text input)
- Right boundary: & (dropdown)
- Parameter Prefix: AribaBuyerGrandChildID (text input)
- Match Case
- Use '#' for any digit
- Advanced... (button)

- 2 Select a type of action: link or form action, cookie, all body, form field, or web_reg_add_cookie.
- 3 For the first three types, specify boundaries of the data in the **Left Boundary** and **Right Boundary** boxes.
- 4 For form field type actions, specify the field name.

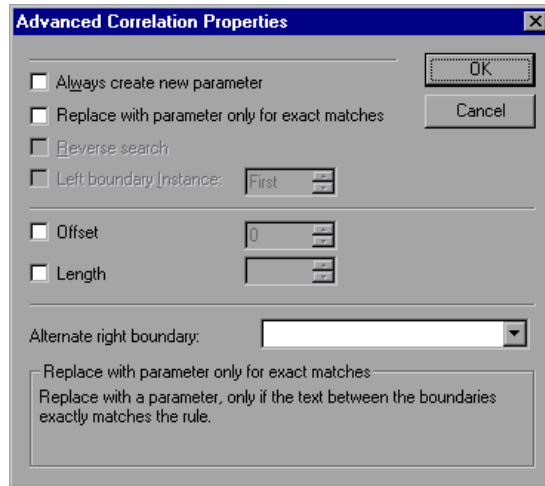


The screenshot shows a dialog box with the following fields and options:

- Action: Parameterize form field value (dropdown)
- Field Name: BV_SessionID (text input)
- Parameter Prefix: AribaBuyerGrandChildID (text input)
- Match Case
- Use '#' for any digit
- Advanced... (button)

- 5 Select the desired options: **Match Case** and/or **Parameter Prefix**. Specify a parameter prefix. To convert all digits to hash signs (#), select **Use # for any digit**.

- 6 To set advanced rules, click **Advanced** in the Correlation node. The Advanced Correlation Properties dialog box opens.

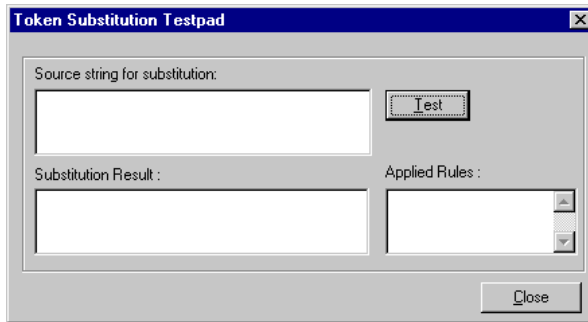


- Select **Always create new parameter** to create a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance.
 - Select **Replace with parameter only for exact matches** to replace a value with a parameter only when the text exactly matches the found value.
 - Select **Reverse Search** to perform a backward search.
 - Select the **Left Boundary Instance** box and specify the desired instance.
 - Select **Offset** to specify an offset for the string within the match.
 - Select **Length** to specify the length of the matched string to save to the parameter. This option may be used in conjunction with the **Offset** option.
 - Specify another right boundary in the **Alternate right boundary** box or select **End of String** or **NewLine Character** from the drop-down menu.
- 7 Click **Test Rule** to test the rule you just defined. For information, see "Testing Rules" on page 642.
- 8 Click **OK** to save the rules and close the dialog box.

Testing Rules

This section applies to user-defined rules that you created for a server with a known context. After you define a new rule in the Correlation Rule dialog box, you can test it before recording your session by applying the rules to a sample string. You test the rules in the Token Substitution Testpad. To use the testpad:

- 1 Select a rule from the left pane and click **Test**. The Token Substitution Testpad dialog box opens.



- 2 Enter text in the **Source String for Substitution** box.
- 3 Click **Test**.

If substitution occurred, you will see the parameterized source text in the **Substitution Result** box and a list of rules that were applied to it in the **Applied Rules** box.

43

Web (HTTP/HTML) -Correlation After Recording

When correlation was not performed during recording, VuGen's built-in Web Correlation mechanism allows you to correlate Vuser scripts after a recording session.

This chapter includes:

- About Correlating with Snapshots on page 644
- Viewing the Correlation Results Tab on page 645
- Setting Up VuGen for Correlations on page 648
- Performing a Scan for Correlations on page 651
- Performing Manual Correlation on page 655
- Defining a Dynamic String's Boundaries on page 660

The following information applies only to Web, Wireless, SAP-Web, and Siebel Web Vuser scripts.

About Correlating with Snapshots

VuGen provides several correlation mechanisms for Web Vuser scripts. The automatic method discussed in Chapter 42, "Web (HTTP/HTML) Correlation Rules" detects dynamic values during recording and allows you to correlate them right away. If you disabled automatic correlation, or if the automatic method did not detect all of the differences, you can use VuGen's built-in correlation mechanism, described in this chapter, to find differences and correlate the values. You can also use this mechanism for scripts that were only partially correlated.

The correlation mechanism uses snapshots to track the results of script execution. Snapshots are graphical representations of Web pages. VuGen captures snapshots of the Web pages during record and replay. You compare the recorded snapshot to any of the replay snapshots to determine which values you need to correlate to successfully run the script. For more information about Record and Replay snapshots, see Chapter 2, "Introducing Service TestVuGen" in *Volume I-Using VuGen*.

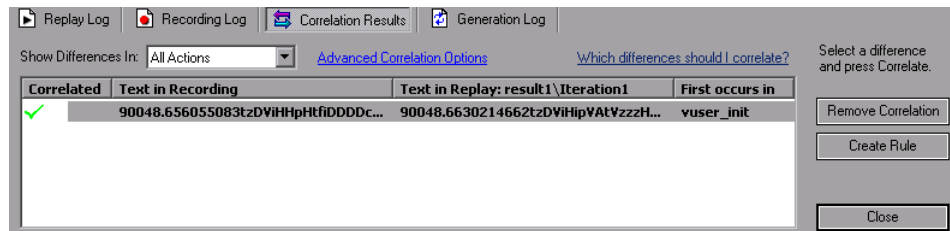
The Web correlation mechanism has a built-in comparison utility that allows you to view the text or binary differences between the snapshots. You can then correlate the differences one-by-one or all at once.

If VuGen's correlation mechanisms are insufficient, or for protocols that do not support these mechanisms, such as Wireless, use manual correlation. For more information, see "Performing Manual Correlation" on page 655.

Viewing the Correlation Results Tab

The Correlation Results tab displays the differences between the Record and Replay snapshots.

When you instruct VuGen to scan the script for correlations, it opens the Output window and displays the differences between the recording and replayed snapshots in the **Correlation Results** tab.



You can display all the differences in the script or only those for the current step or action, by selecting the desired option from the **Show Differences In** list box.

Differences that were correlated are indicated by a check mark in the **Correlated** column. The next two columns, **Text in Recording**, **Text in Replay** show the text differences between the snapshots. The next column, **First occurs in**, indicates the Action in which the correlation was first detected.

After you detect the differences between the snapshots, you correlate them one at a time by selecting the correlation and clicking **Correlate**. VuGen also allows you to undo a specific correlation using the **Remove Correlation** button. If you expect one of the detected correlations to occur in subsequent recordings, you can create a new correlation rule. By creating rules, you enable VuGen to recognize differences during recording and automatically correlate them. For more information, see "Creating a Rule" on page 646.

When you correlate a value using the this mechanism, VuGen inserts a `web_reg_save_param` function and a comment into your script indicating that a correlation was done for the parameter. It also indicates the original value.

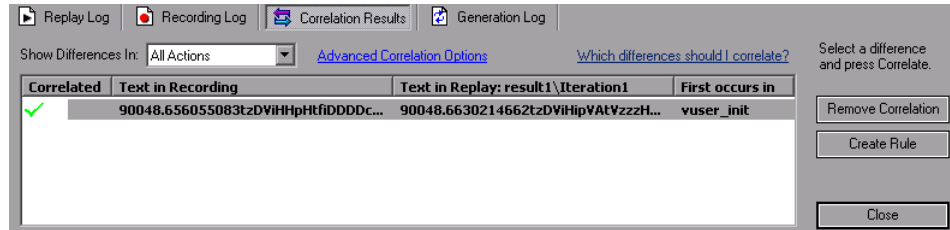
```
// [WCSPARAM WCSParam_Diff1 14 reserveFlights] Parameter {WCSParam_Diff1}
created by Correlation Studio
  web_reg_save_param("WCSParam_Diff1", "LB= NAME=\"", "RB=\"", "Ord=5",
"Search=Body", "RelFrameId=1", LAST);
  web_submit_form("reservations.pl",
    "Snapshot=t4.inf",
    ITEMDATA,
    "Name=depart", "Value=Denver", ENDITEM,
    "Name=departDate", "Value=06/25/2004", ENDITEM,
    "Name=arrive", "Value=Los Angeles", ENDITEM,
    "Name=returnDate", "Value=06/26/2004", ENDITEM,
    "Name=numPassengers", "Value=1", ENDITEM,
    "Name=roundtrip", "Value=<OFF>", ENDITEM,
    "Name=seatPref", "Value=None", ENDITEM,
    "Name=seatType", "Value=Coach", ENDITEM,
    "Name=findFlights.x", "Value=44", ENDITEM,
    "Name=findFlights.y", "Value=12", ENDITEM,
    LAST);
  lr_think_time(12);
```

Creating a Rule

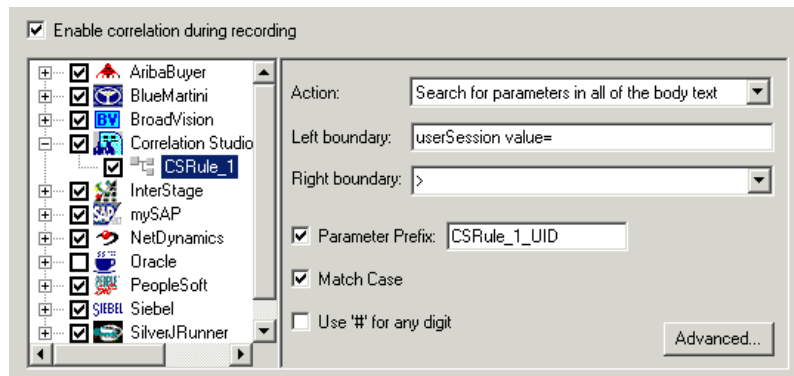
You can create a rule directly from the list of Correlated Results. Creating a rule, enables VuGen to recognize the difference during recording and automatically correlate it.

To create a rule from one of the detected correlations:

Select the correlation and click **Create Rule**. You can also create a rule by selecting a correlation and choosing **Create Correlation Rule** from the right-click menu.

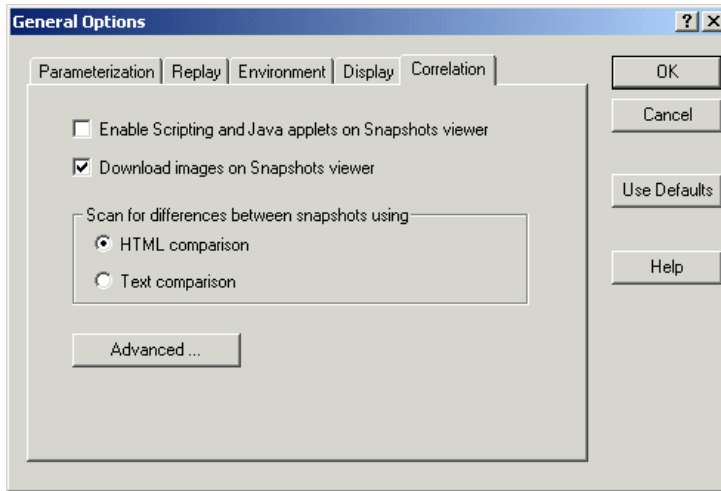


VuGen adds this rule to the list of **Correlation** rules. You can view this rule in the Recording Options Correlation node. In the following example, VuGen added the rule as CSRule_1.



Setting Up VuGen for Correlations

You set the global Correlation setting under the General options. These options instruct the Users to save correlation information during replay, to be used at a later stage. You can specify the type of comparison to perform when comparing snapshots: HTML or text. In the Advanced options, you can indicate which characters should be treated as delimiters.



- ▶ **Enable Scripting and Java applets on Snapshots viewer.** Allows VuGen to run applets and JavaScript in the snapshot window. This is disabled by default because it uses a lot of resources.
- ▶ **Download images on Snapshots viewer.** Instructs VuGen to display graphics in the Snapshot view. If you find that the displaying of images in the viewer is very slow, you can disable this option. This option is enabled by default.
- ▶ **Scan for differences between snapshots using.** Select a comparison method:
 - ▶ **HTML Comparison.** Only display the differences in HTML code.
 - ▶ **Text Comparison.** Display all text, HTML, and binary differences.

Note: In most cases, we recommend that you work with the default HTML comparison method. If your script contains non-HTML tags, you can use the Text comparison method.

- **Advanced.** Opens the Advanced Correlation dialog box.

Advanced Correlation dialog box

This dialog box lets you specify the characters to be treated as delimiters.

- **Characters that should be treated as delimiters.** Specifies one or more non-standard delimiters.
- **Additional Delimiters.** You can specify standard delimiters such as Carriage Return, New line and Tab characters. To change this setting, clear the check box next to the delimiter.
- **Ignore differences shorter than ... characters.** Allows you to specify a threshold for performing correlation. When VuGen compares the recorded data with the replay data during the scanning process, it detects differences. It will not correlate the differences unless the number of different characters is greater than or equal to the threshold value. The default value is 4 characters.
- **Issue a warning for large correlations.** Issues a warning if you try to correlate a string whose size is 10 KB or larger.

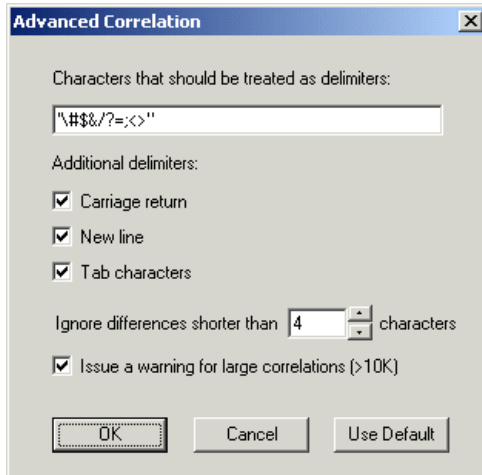
Setting the Correlation Preferences

Before recording a session, you configure the correlation preferences.

To set the correlation preferences:

- 1** Select **Options > General** and select the **Correlation** tab.
- 2** Select **Enable Scripting and Java applets on Snapshots viewer** to allow VuGen to run applets and JavaScript in the snapshot window.

- 3 To instruct VuGen to display graphics in the Snapshot view, select the **Download images on Snapshots viewer** option.
- 4 Select the comparison method: **HTML comparison** or **Text Comparison** (for non-HTML elements only).
- 5 To set the delimiter characters, click **Advanced** to open the Advanced Correlation dialog box.



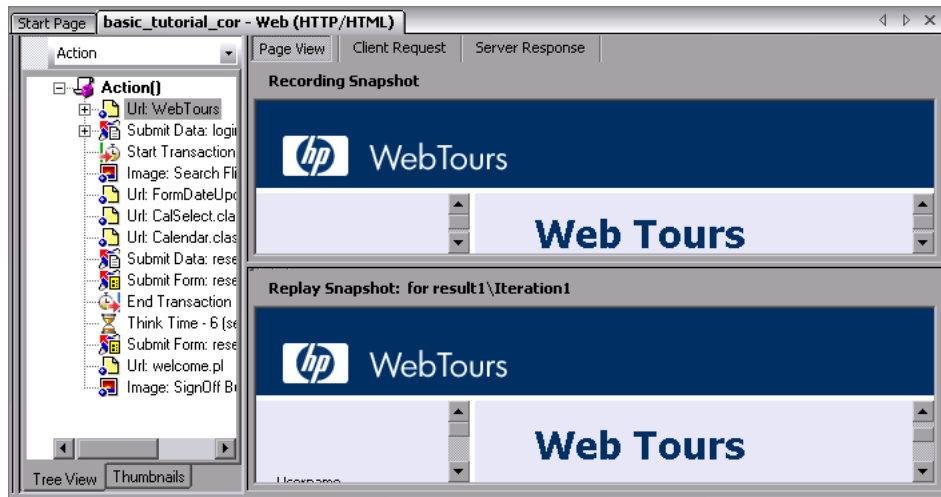
- 6 In the **Characters that should be treated as delimiters** box, specify all characters that are to be treated as delimiters.
- 7 Select the desired options in the **Additional delimiters** section, to specify one or more standard delimiters.
- 8 Specify a threshold for the correlation in the **Ignore differences shorter than** box. When VuGen compares the recorded data with the replay data during the scanning process, it detects differences. It will not correlate the differences unless the number of different characters is greater than or equal to the threshold value.
- 9 To **issue a warning for large correlations**, select the option's check box.
- 10 Click **OK** to accept the Advanced Correlation settings and close the dialog box.
- 11 Click **OK** in the General Options dialog box to accept the Correlation setting and close the dialog box.

Performing a Scan for Correlations

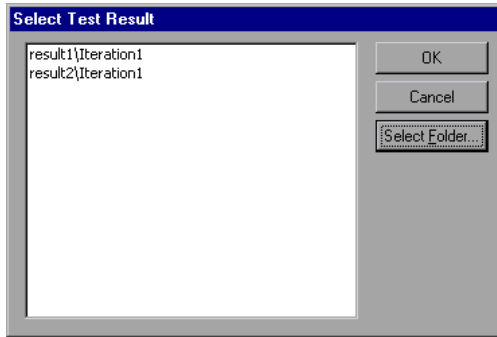
You can use VuGen's snapshot window to determine which values within your script are dynamic and require correlation. The following section describes how to automatically scan the script for differences and use VuGen to perform the necessary correlations.

To scan your script for correlations:

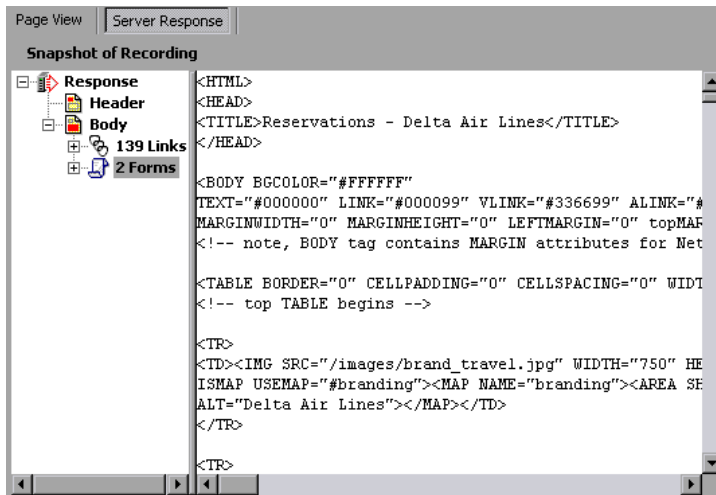
- 1** Open a script and view it in Tree view (**View > Tree View**). Display the snapshots (**View > Snapshot > View Snapshot**).
- 2** Select a script step in the Tree view from the left pane. A snapshot opens in the right pane.
- 3** To display both the recording snapshot and the first replay snapshot, click **View > Snapshot > Recorded and Replayed**.



- 4 To use a snapshot other than the first, click **View > Snapshot > Select Iteration**. A dialog box opens, displaying the folders that contain snapshot files. These are usually the **result** and **Iteration** folders below the script's folder.

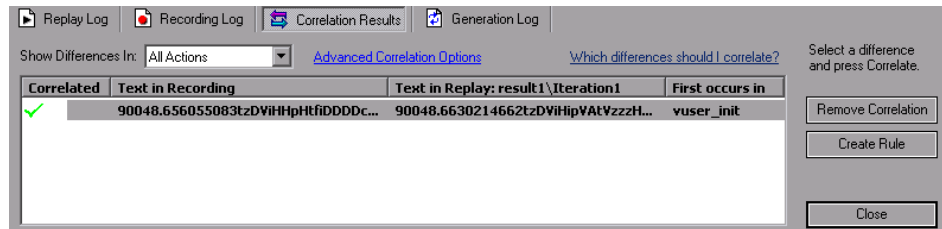


- 5 To select a snapshot file in a folder other than the one in the subfolders of the script, click **Select Folder**. Browse to the desired location, and click **OK**.
- 6 To view the HTML code, click the **Server Response** tab. Expand the Body branch.



To return to the page view, click the **Page View** tab.

- 7 Select **Vuser > Scan for Correlations** or click the **Find Correlations** button. VuGen scans the script for dynamic values that need to be correlated and displays them in the Correlation Results tab.



- 8 View all differences or select a filter method in the **Show Differences In** list box. The options are **All Actions**, **Current Action**, or **Current Step Only**.

Determining the Differences to Correlate

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

The following strings most probably require correlation:

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

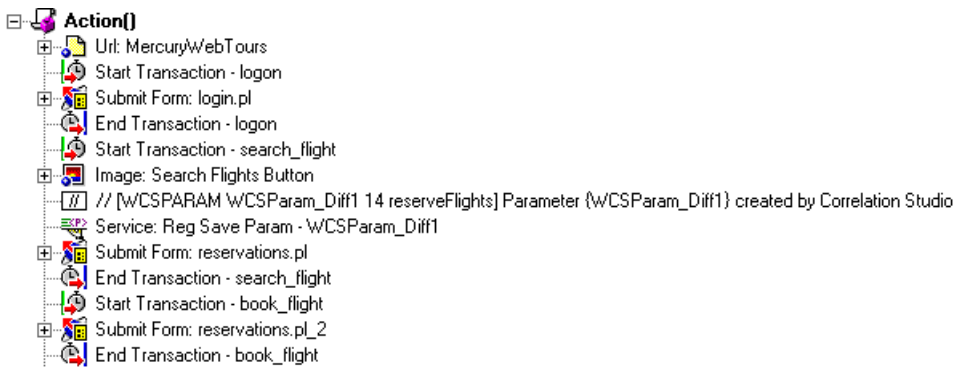
If you are in doubt whether a difference should be correlated, correlate only that difference and then run your script. Check the Replay log to see if the issue was resolved.

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

Once you determine that a difference needs to be correlated, you instruct VuGen to correlate it.

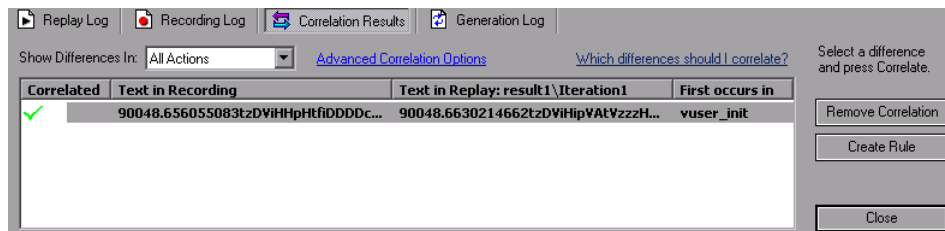
To correlate the differences:

- 1 View the differences in the Correlation tab, and select the one you want to correlate. We recommend that you correlate only one difference at a time.
- 2 Click **Correlate**. VuGen places a green check mark next to differences that were correlated and inserts a `web_reg_save_param` function into the script.

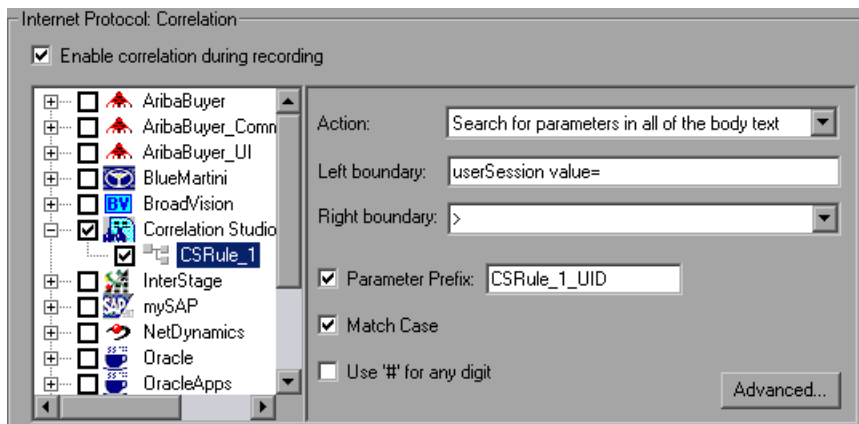


Repeat this step for all differences you want to correlate.

- 3 To create a rule from one of the detected correlations, select the correlation and click **Create Rule**. This is also available from the right-click menu. VuGen issues a message confirming that your rule was created.



To view this rule, open the Recording Options (CTRL +F7) and select the **Correlation** node. Expand the **Correlation Studio** entry and select your rule.



4 To undo a correlation, select the difference and click **Remove Correlation**.

5 Select **File > Save** to save the changes to your script.

Performing Manual Correlation

For Web Vusers, VuGen's automatic or rule-based correlation usually correlates the scripts dynamic functions so that you can run the script successfully. You can also perform correlation after the recording session using VuGen's snapshot comparison.

For Wireless Vusers and other Vuser scripts for which automatic correlation did not apply, VuGen also allows you to manually correlate your scripts. You manually correlate a script by adding the code correlation functions. The function that allows you to dynamically save data to a parameter is **web_reg_save_param**.

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

The function's syntax is as follows:

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

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

NotFound	The handling method when a boundary is not found and an empty string is generated. "ERROR," the default, indicates that VuGen should issue an error when a boundary is not found. When set to "EMPTY," no error message is issued and script execution continues. Note that if Continue on Error is enabled for the script, then even when NOTFOUND is set to "ERROR," the script continues when the boundary is not found, but it writes an error message to the Extended log file.
LB	The left boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.
RB	The right boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.
RelFrameID	The hierarchy level of the HTML page relative to the requested URL. The possible values are ALL or a number.
Search	The scope of the search—where to search for the delimited data. The possible values are Headers (search only the headers), Body (search only Body data, not headers), or ALL (search Body and headers). The default value is ALL.
ORD	This optional parameter indicates the ordinal or occurrence number of the match. The default ordinal is 1. If you specify "All," it saves the parameter values in an array.

SaveOffset	The offset of a sub-string of the found value, to save to the parameter. The default is 0. The offset value must be non-negative.
Savelen	The length of a sub-string of the found value, from the specified offset, to save to the parameter. The default is -1, indicating until the end of the string.
Convert	The conversion method to apply to the data: HTML_TO_URL: convert HTML-encoded data to a URL-encoded data format HTML_TO_TEXT: convert HTML-encoded data to plain text format

To manually correlate your script:

- 1 Identify the statement that contains dynamic data and the patterns that characterize the boundaries of the data. See "Defining a Dynamic String's Boundaries" on page 660.
- 2 In the script, replace the dynamic data with your own parameter name. See below for more details.
- 3 Add the `web_reg_save_param` function into the script before the statement that contains the dynamic data. See "Adding a Correlation Function" on page 658 or the *Online Function Reference* (**Help > Function Reference**).

Replacing Dynamic Data with a Parameter

Identify the actual dynamic data in the recorded statement, then search the entire script for the dynamic data and replace it with a parameter. Give the parameter any name and enclose it with braces: {param_name}. You can include a maximum of 64 parameters per script.

To replace dynamic data with a parameter:

Select **Edit > Replace** from the VuGen main window to display the Search and Replace dialog box. Search the entire script for the dynamic data and replace it with a parameter.

Adding a Correlation Function

You insert the **web_req_save_param** statement to save dynamic data in a script. This function tells VuGen to create a parameter that saves the runtime value of the dynamic data during replay.

When you run the script, the **web_req_save_param** function scans the subsequent HTML page that is accessed. It searches for an occurrence of the left boundary, followed by any string, followed by the right boundary. When such an occurrence is found, VuGen assigns the string between the left and right boundaries to the parameter named in the function's argument. After finding the specified number of occurrences, **web_req_save_param** does not search any more HTML pages. The Vuser continues with the next step in the script.

Sample Correlation for Web Vusers

Suppose the script contains a dynamic session ID:

```
web_url("FirstTimeVisitors",  
  "URL=/exec/obidos/subst/help/first-time-visitors.html/002-8481703-  
  4784428>Buy books for a penny ",  
  "TargetFrame=",  
  "RecContentType=text/html",  
  "SupportFrames=0",  
  LAST);
```

You insert a **web_req_save_param** statement before the above statement:

```
web_req_save_param ("user_access_number", "NOTFOUND=ERROR", "LB=first-  
time-visitors.html/", "RB=>Buy books for a penny", "ORD=6", LAST);
```

After implementing correlated statements, the modified script looks like this, where `user_access_number` is the name of the parameter representing the dynamic data.

```
web_url("FirstTimeVisitors",
  "URL=/exec/obidos/subst/help/first-time-  

  visitors.html/{user_access_number}Buy books for a penny",
  "TargetFrame=",
  "RecContentType=text/html",
  "SupportFrames=0",
  LAST);
```

Note: Each correlation function retrieves dynamic data once, for the subsequent HTTP request. If another HTTP request at a later point in the script generates new dynamic data, you must insert another correlation function.

Sample Correlation for Wireless Users

Suppose your script contains a dynamic session ID for a WAP connection:

```
web_url("login.po;sk=luZSuuRIHUMnpF-wpK8PzEpy(1YOSBSMy)",
  "URL=http://room33.com/portal/login.po;sk=luZSuuRIHUMnpF-  

  wpK8PzEpy(1YOSBSMy)",
  "Resource=0",
  "RecContentType=text/vnd.wap.wml",
  "Mode=HTML",
  LAST);
```

You insert a `web_reg_save_param` statement before the above statement and replace the dynamic value with the parameter. In the following example, the `web_reg_save_param` functions saves the login ID string to a variable called SK. It saves binary data, denoted by the RB/BIN attribute, and sets the left boundary as "sk=".

```
web_reg_save_param(  
    "SK",  
    "LB=sk=",  
    "RB/BIN=#login\\x00\\x01\\x03",  
    "Ord=1",  
    LAST);  
  
web_url("login.po;sk={SK}",  
    "URL=http://room33.com/portal/login.po;sk={SK}",  
    "Resource=0",  
    "RecContentType=text/vnd.wap.wml",  
    "Mode=HTML",  
    LAST);
```

Defining a Dynamic String's Boundaries

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

- ▶ Always analyze the location of the dynamic data within the HTML code itself, and not in the recorded script.
- ▶ Identify the string that is immediately to the left of the dynamic data. This string defines the left boundary of the dynamic data.
- ▶ Identify the string that is immediately to the right of the dynamic data. This string defines the right boundary of the dynamic data.

- **web_reg_save_param** looks for the characters between (but not including) the specified boundaries and saves the information beginning one byte after the left boundary and ending one byte before the right boundary. **web_reg_save_param** does not support embedded boundary characters.

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

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

```
web_set_max_html_param_len("1024");
```


44

Web (HTTP/HTML) - Handling XML Pages

VuGen's Web Vusers support Web pages containing XML code.

This chapter includes:

- About Testing XML Pages on page 663
- Viewing XML as URL Steps on page 664
- Inserting XML as a Custom Request on page 667
- Viewing XML Custom Request Steps on page 668

About Testing XML Pages

VuGen supports record and replay for XML code within Web pages.

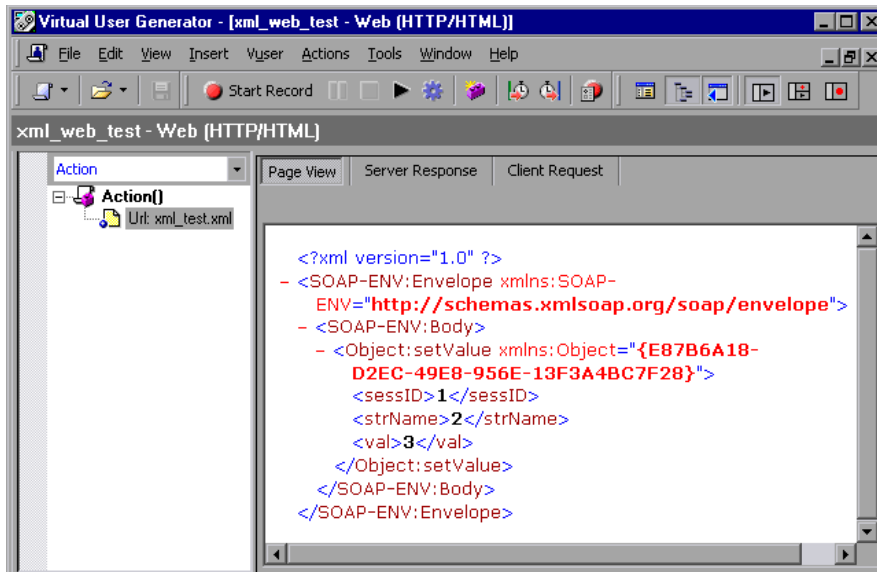
The XML code can appear in the script as a regular URL step or as a custom request. VuGen detects the HTML and allows you to view each document type definition (DTD), its entities, and its attributes. VuGen can interpret the XML when the MIME type displayed in the **RecContentType** attribute or the MIME type returned by the server during replay, ends with **xml**, such as **application/xml** or **text/xml**. The DTD is color coded, allowing you to identify each one of the elements. You can also expand and collapse the tree view of the DTD.

When you expand the DTD, you can parameterize the attribute values. You can also save the values in order to perform correlation using the standard correlation functions. For more information about the correlation functions, see the *Online Function Reference* (**Help > Function Reference**).

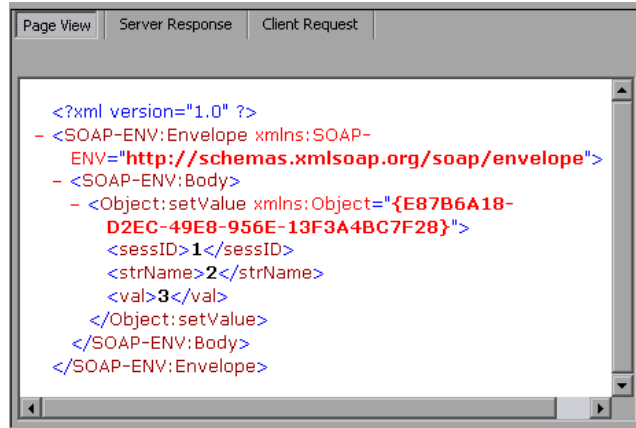
Note: VuGen cannot display a DTD with **XML islands**, segments of XML embedded inside an HTML page. VuGen only displays pages that are entirely XML.

Viewing XML as URL Steps

One way to test a page with XML code, is to record it with VuGen. You record the XML pages as you would record a standard Web page. VuGen records the DTD and all of the XML elements. It does not create a snapshot for the XML page. Instead, for each XML step it displays the XML code in the snapshot frame under the Server Response tab.



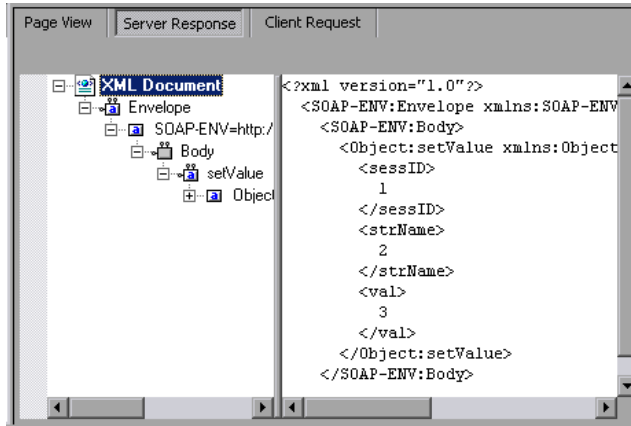
VuGen creates a color-coded expandable hierarchy of the DTD in the snapshot frame. Click on the "+" to expand an item, and click on the "-" to collapse it. VuGen displays all XML tags in brown, and values in black.



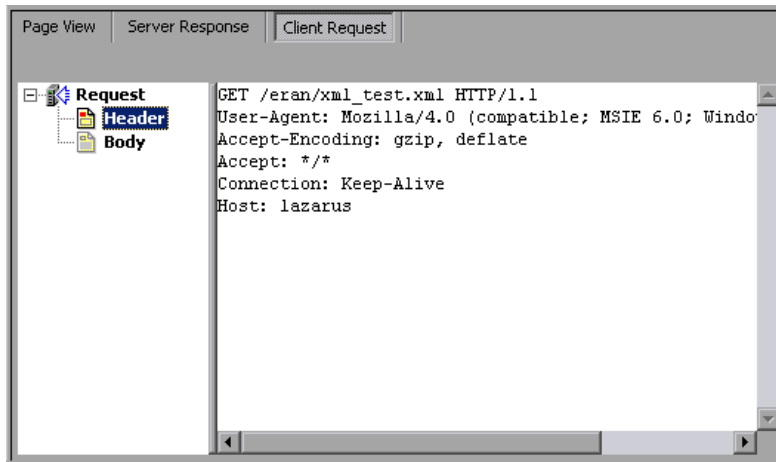
```
<?xml version="1.0" ?>
- <SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope">
- <SOAP-ENV:Body>
- <Object:setValue xmlns:Object="{E87B6A18-
  D2EC-49E8-956E-13F3A4BC7F28}">
  <sessID>1</sessID>
  <strName>2</strName>
  <val>3</val>
  </Object:setValue>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

To replace any of the constant values with a parameter, select a value, perform a right-click, and select **Replace with a Parameter**. Follow the standard procedure for parameterization. For more information, see "Creating Parameters" in *Volume I-Using VuGen*.

You can also view the Server response and Client request for the XML page by clicking the appropriate tab. The following example shows the Server response of an XML page. Note that you can expand and collapse all branches of the XML tree.



The following example shows the Client Request for the header of an XML page:

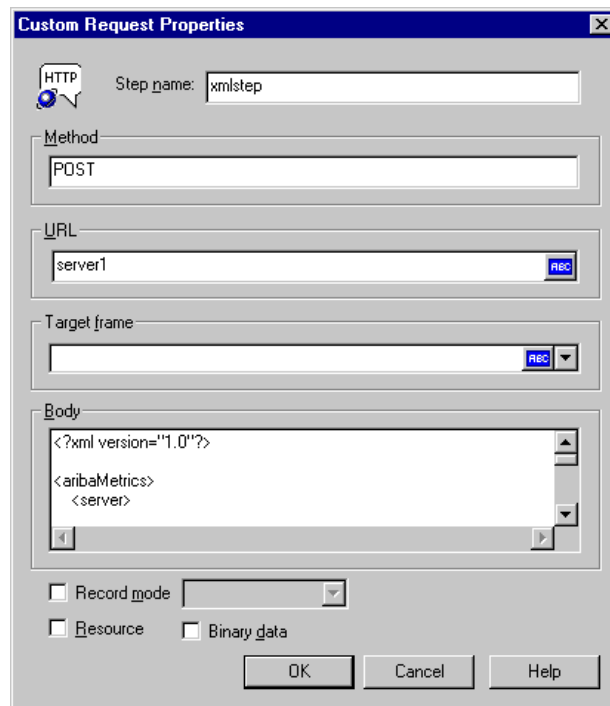


Inserting XML as a Custom Request

You can also test your XML pages by inserting the XML code as a custom request. In this mode, the **Custom Request** properties box displays the elements of the DTD in either text or XML format.

To add XML code as a Custom Request:

- 1 View the script in tree view mode, place the cursor at the desired location, and select **Insert > Add Step**. The Add Step dialog box opens.
- 2 Scroll to the bottom of the list and select **Custom Request**. Click **OK**. The Custom Request Properties dialog box opens.
- 3 Enter a step name, method (GET or POST), URL, and target frame (optional).
- 4 Copy the XML code from your browser or editor and paste it into the **Body** section of the Custom Request Properties box.



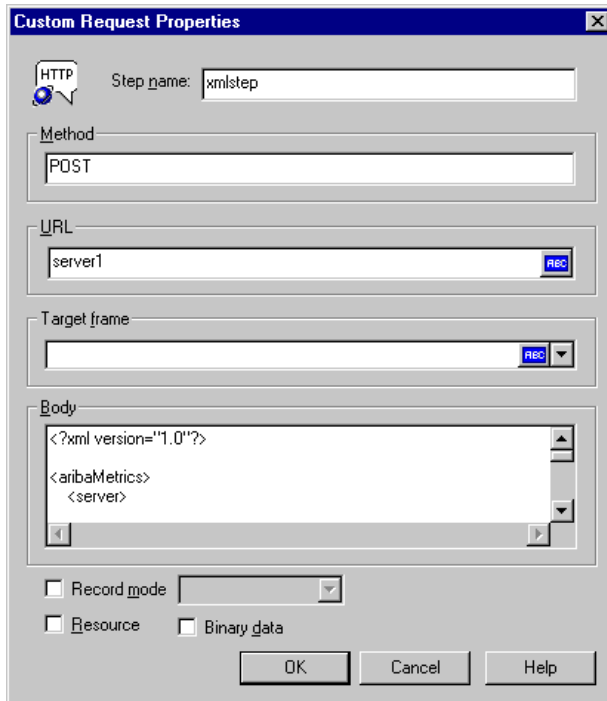
- 5 Select the applicable replay options: **Record mode**, **Resource**, or **Binary data**. For more information, see Chapter 41, "Modifying Web and Wireless Vuser Scripts."
- 6 Click **OK**. VuGen places the custom request step into your script.

Viewing XML Custom Request Steps

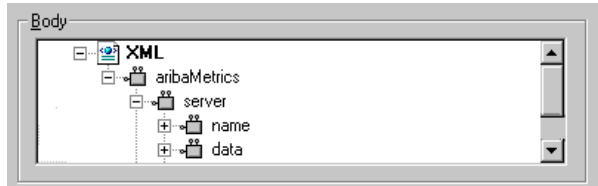
You can view or modify the XML code implemented as a custom request step, at any time. VuGen provides a viewer that allows you to view the hierarchy of the DTD, and expand and collapse the elements as needed.

To view the XML code of a custom request step:

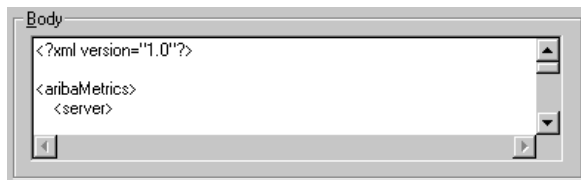
- 1 View the script in tree view mode, and select the desired step.
- 2 Select **Properties** from the right-click menu. The Custom Request Properties dialog box opens.



The bottom section of the dialog box displays the XML code. If the **RecContentType** attribute is set to **text/xml**, by default VuGen displays the code in an XML format hierarchy. In this mode, the XML code is not editable.

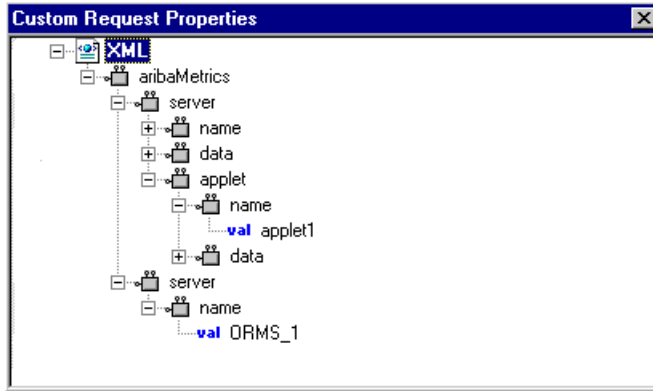


If the **RecContentType** attribute is set to any type other than **text/xml**, VuGen displays the code in plain text format. In this mode, the XML code is editable.



- 3 To switch between the text and XML views, select **XML view** or **Text view** from the right-click menu.

- 4 When you are in XML view, you can view the code in a larger window. Select **Extended view** from the right-click menu. To switch back to the dialog box view, select **Normal view** from the right-click menu.



45

Oracle NCA Protocol

You can use VuGen to create scripts that emulate an Oracle NCA user. You record typical NCA business processes with VuGen. You then run the script to emulate users interacting with your system.

This chapter includes:

- About Creating Oracle NCA Vuser Scripts on page 672
- Getting Started with Oracle NCA Vusers on page 673
- Recording Guidelines on page 674
- Enabling the Recording of Objects by Name on page 676
- Oracle Applications via the Personal Home Page on page 679
- Using Oracle NCA Vuser Functions on page 681
- Understanding Oracle NCA Vusers on page 681
- Testing Oracle NCA Applications on page 683
- Correlating Oracle NCA Statements for Load Balancing on page 686
- Additional Recommended Correlations on page 687
- Recording in Pragma Mode on page 689

About Creating Oracle NCA Vuser Scripts

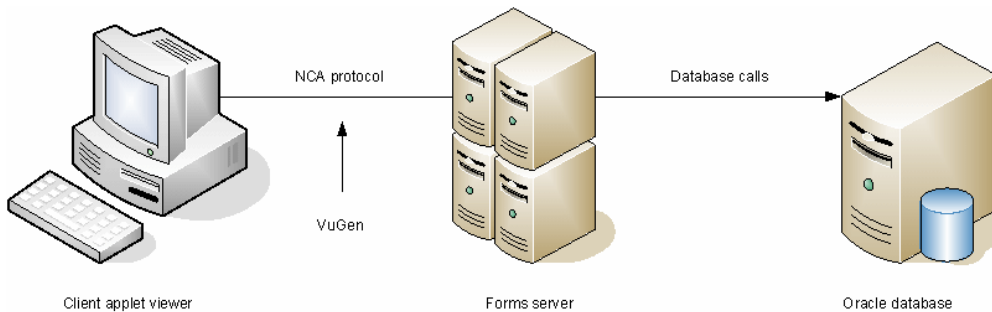
Oracle NCA is a Java-based database protocol. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer.

This eliminates the need for client software and allows you to perform database actions from all platforms that support the applet viewer. There is a Vuser type specifically designed to emulate an Oracle NCA client.

The NCA environment is a three-tier environment. The user first sends an http call from his browser to a Web server. This call accesses the startup HTML page which invokes the Oracle Applications applet. The applet runs locally on the client machine—all subsequent calls are communicated between the client and the Forms server through the proprietary NCA protocol.

The client (applet viewer) communicates with the application server (Oracle Forms server) which then submits information to the database server.

VuGen records and replays the NCA communication between the client and the Forms server (application server).



When you record an Oracle NCA session, VuGen records all of the NCA and Web actions, even if you only created a single protocol script. If you know in advance that the Web functions are important for your test, create a multi-protocol script from the beginning for the Oracle NCA and Web protocols.

If you initially created a single protocol script for Oracle NCA, and at a later stage you require the Web functions for testing, you can regenerate your script in VuGen to add the Web functions, without having to re-record the session. You indicate this from the Protocols node in the Regenerate Script dialog box. For more information, see "Recording with VuGen" in *Volume I-Using VuGen*.

Getting Started with Oracle NCA Vusers

The following procedure outlines how to create an Oracle NCA Vuser script.

1 Make sure that the recording machine is properly configured.

Make sure that your machine is configured to run the Oracle NCA applet viewer, before you start VuGen. You must also make sure VuGen supports your version of Oracle Forms. For more information, see "Recording Guidelines" on page 674 and the Readme file.

2 Create a skeleton Oracle NCA Vuser script.

Use VuGen to create a skeleton Oracle NCA Vuser script. For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

3 Record typical user actions.

Begin recording, and perform typical actions and business processes from the applet viewer. VuGen records your actions and generates a Vuser script. For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

4 Enhance the Vuser script.

Use the Insert menu to add transactions, rendezvous points, comments, and messages in order to enhance the Vuser script. For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

5 Parameterize the script.

Replace recorded constants with parameters. For details, see "Creating Parameters" in *Volume I-Using VuGen*.

6 Set the run-time properties for the script.

Configure run-time settings for the Vuser script. The run-time settings define certain aspects of the script execution. For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

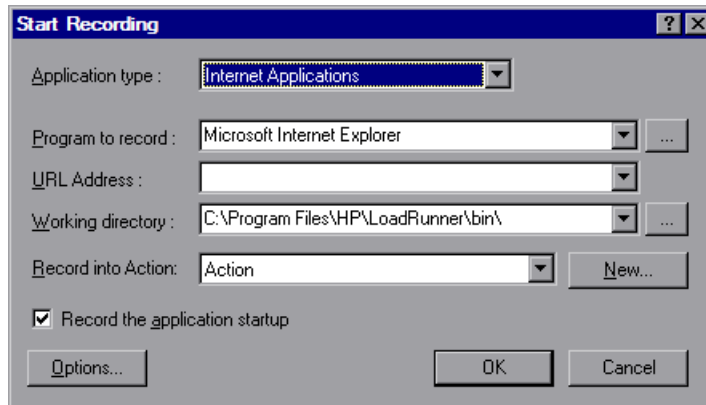
7 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

Recording Guidelines

When recording an Oracle NCA Vuser script, follow these guidelines:

- Specify which browser VuGen should use when recording an Oracle NCA session. In the Start Recording dialog box, select the desired browser in the **Program to Record** list. The list contains all of the available browsers.



- Close all browsers before you begin recording.

- Record the login procedure in the `vuser_init` section. Record a typical business process in the Actions section. When you run the script, you can then specify multiple iterations for a specific business process. For more information, see Chapter 5, "Recording with VuGen" in *Volume I-Using VuGen*.

```
vuser_init()
{
    nca_set_connect_opt(SCALE_INFO, 11, 18);
    nca_connect_server("labm1orcl05.devlab.ad", "9000",
"module=/opt/applvis/visappl/fnd/11.5.0/forms/US/FNDSCSGN
userid=APPLSYSPUB/PUB@VIS fndnam=APPS record=names ");
    nca_set_window("Oracle Applications");
    nca_edit_set("SIGNON_USERNAME_0", "OPERATIONS");
    nca_obj_type("SIGNON_USERNAME_0", '\t', 0);
    nca_edit_set("SIGNON_PASSWORD_0", lr_decrypt("4768d647f4f1840f2e46d5"));
    nca_button_press("SIGNON_CONNECT_BUTTON_0");
    return 0;
}
```

- Due to a Netscape limitation, you cannot launch an Oracle NCA session within Netscape when another Netscape browser is already running on the machine.
- VuGen supports the recording of Oracle Forms applications using the Forms Listener Servlet in multi-protocol mode. The application server uses the **Forms Listener Servlet** to create a runtime process for each client. The runtime process, **Forms Server Runtime**, maintains a persistent connection with the client and sends information to and from the server.

To support Forms 4.5 in replay, set the following in the **mdrv.dat** file:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp110.dll
WIN95_EXT_LIBS=ncarp110.dll
LINUX_EXT_LIBS=liboranca.so
SOLARIS_EXT_LIBS=liboranca.so
HPUX_EXT_LIBS=liboranca.sl
AIX_EXT_LIBS=liboranca.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api
```

To restore Forms support for versions later than 4.5, restore the original values.

Enabling the Recording of Objects by Name

When recording an Oracle NCA script, you must record the session using object names instead of the standard object ID. If the script is recorded using the object ID, replay will fail because the ID is generated dynamically by the server and differs between record and replay. You can verify that your script is being recorded with object names by examining the **nca_connect_server** statement.

```
nca_connect_server("199.35.107.119","9002"/*version=11i*/, "module=/d1/oracle/visap
pl/fnd/11.5.0/forms/US/FNDSCSGN userid=APPLSYSPUB/PUB@VIS fndnam=apps
record=names ");
```

If the **record=names** argument does not appear in the **nca_connect_server** function, you are recording object IDs. You can instruct VuGen to record object names in by modifying one of the following:

- ▶ Startup HTML File
- ▶ URL to Record
- ▶ Forms Configuration File

Note that the ability to capture the developer name for all objects was introduced in Oracle Forms6i Patch 9 (Oracle Forms Version: 6.0.8.18.3). Test Starter Kit scripts that were written before the release of Oracle Forms 6i Patch 9 will not have the developer name as part of an object's physical description, except for the edit fields.

Startup HTML File

If you have access to the startup HTML file, you instruct VuGen to record object names instead of its object ID by setting the **record=names** flag in the startup file, the file that is loaded when you start the Oracle NCA application.

Edit the startup file that is called when the applet viewer begins. Modify the line:

```
<PARAM name="serverArgs ... fndnam=APPS">
```

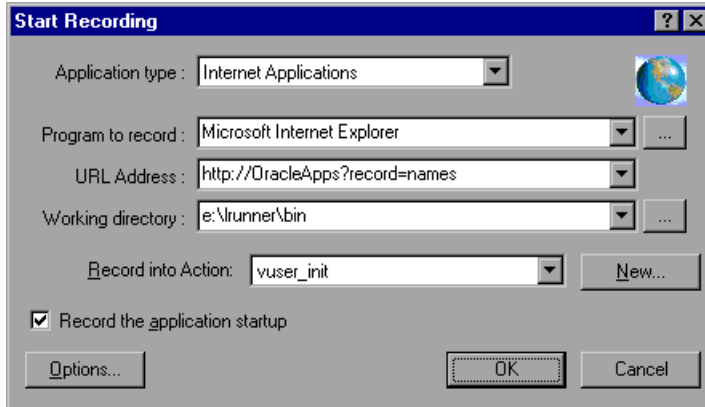
and add the Oracle key "record=names":

```
<PARAM name="serverArgs ... fndnam=APPS record=names">
```

URL to Record

If you do not have access to the startup HTML file, you can still have Oracle NCA record object names instead of its object ID by modifying the URL to record. The following solution only works if the startup HTML file does not reference another file while loading.

For this solution, you add "**?record=names**" after the URL in the Start Recording dialog box, after the URL name to record. This allows VuGen to record object names for the session.



Forms Configuration File

If the application has a startup HTML file that references a Forms Web CGI configuration file **formsweb.cfg** (a common reference), you may encounter problems if you add **record=names** to the Startup file.

In this situation, you should modify the configuration file.

To modify the configuration file to record object names:

- 1 Go to the Forms Web CGI configuration file.
- 2 Define a new parameter in this file (see sample Web CGI configuration file below for this change).

```
serverApp=forecast
serverPort=9001
serverHost=easgdev1.dats.ml.com
connectMode=socket
archive=f60web.jar
archive_ie=f60all.cab
xrecord=names
```

- 3 Open the startup HTML file and locate PARAM NAME="serverArgs".

- 4 Add the variable name as an argument to the `ServerArgs` parameter, for example, `record=%xrecord%`.

```
<PARAM NAME="serverArgs" VALUE="module=%form% userid=%userid%
%otherParams% record=%xrecord%">
```

- 5 Alternatively, you can edit the `basejini.htm` file in Oracle Forms installation directory. This file is the default HTML file for running a form on the web using JInitiator-style tags to include the Forms applet. In the `basejiniin.hmt` file add the following line to the parameter definitions:

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the `<EMBED>` tag, add the following line:

```
serverApp="%serverApp%"
logo="%logo%"
imageBase="%imageBase%"
formsMessageListener="%formsMessageListener%"
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file `formsweb.cfg`, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the `basejini.htm` file and store it at another location. In the servlet configuration file, edit the `baseHTMLJinitiator` parameter to point to the new file.

Oracle Applications via the Personal Home Page

When launching Oracle Forms 6i applications by logging in through the **Personal Home Page**, you must set several system profile options at the user level. It is desirable to pass such variables at the user level, and not at the site level, where it will affect all users.

To configure the "ICX: Forms Launcher" profile:

- 1 Sign on to the application and select the "System Administrator" responsibility.

- 2** Select **Profile/System** from the Navigator menu.
- 3** Within the **Find System Profile Values** form:
 - a** Select the **Display:Site** option
 - b** **Users** = <your user logon> (i.e. operations, mfg, and so on)
 - c** **Enter Profile** =%ICX%Launch%
 - d** Click **Find**.
- 4** Update the User value to the **ICX:Forms Launcher** profile:
 - ▶ If no parameter has been passed to the URL, append the following string to the end of the URL of the user value: `?play=&record=names`
 - ▶ If a parameter has been passed to the URL, append the following string to the end of the URL of the user value: `&play=&record=names`
- 5** Save the transaction.
- 6** Log out of the Oracle Forms session.
- 7** Log out of the Personal Home Page session.
- 8** Sign on again via the **Personal Home Page** using your username.

If you were unable to update the ICX: Forms Launcher profile option at the user level, open the **Application Developer** responsibility and select the **Updatable** option for the ICX_FORMS_LAUNCHER profile.

The first parameter passed to the URL, must begin with a question mark (?). You pass all subsequent parameters with an ampersand (&). In most cases, the URL already contains parameters, which you can identify by searching for a question mark.

Using Oracle NCA Vuser Functions

VuGen records typical NCA business processes and generates Oracle NCA-specific functions. The functions use an **nca** prefix.

The NCA functions are divided into the following categories: Button Object, Connection, Combo Box Object, Edit and Edit Box Object, Flexfield Window, Java Object, List Object, Menu Object, Message Object, Object, Response Object, Scroll Object, Session, Tab Object, Tree Object, and Window Object Functions.

You can also manually program any of the functions into your Vuser script. In text view, you can manually add new functions utilizing the Intellisense and Complete Function features. In Tree view, select **Insert > New Step** and select the desired step.

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

You can further enhance your script with C Vuser functions such as **lr_output_message** and **lr_rendezvous**. For information on using these functions, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

Understanding Oracle NCA Vusers

When you create an Oracle NCA Vuser script, VuGen records all of the NCA communication between the client and the application server. While you record, VuGen generates context sensitive functions. These functions describe your actions on the database in terms of GUI objects (such as windows, lists, and buttons). As you record, VuGen inserts the context sensitive functions into the Vuser script.

After you finish recording, you can modify the functions in your script, or add additional functions to enhance it. For information about enhancing Vuser script, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*. For a list of the available Oracle NCA Vuser functions, see "Using Oracle NCA Vuser Functions" on page 681. For details of these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following segment, the user selected an item from a list (**nca_list_activate_item**), pressed a button (**nca_button_press**), retrieved a list value (**nca_lov_retrieve_items**), and performed a click in an edit field (**nca_edit_click**). The logical names of the objects are the parameters of these functions.

```
...
nca_lov_select_item("Responsibilities","General Ledger, Vision Operations");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","+ Journals");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0"," Enter");
nca_button_press("GLXJEENT.TOOLBAR.LIST.0");
nca_lov_find_value("Batches","");
nca_lov_retrieve_items("Batches",1,9);
nca_lov_select_item("Batches","AR 1020 Receivables 2537: A 1020");
nca_edit_click("GLXJEENT.FOLDER_QF.BATCH_NAME.0");
...
```

In certain tests, such as those performed on Oracle Configurator applications, information returned by one function is required throughout the session. VuGen automatically saves the dynamic information to a parameter, by inserting a **web_reg_save_param** function into the script. In the following example, the connection information is saved to a parameter called NCAJServSessionID.

```
web_reg_save_param ("NCAJServSessionId", "LB=\r\n\r\n", "RB=\r",
LAST);
web_url("f60servlet",
"URL=http://ussciforms05.sfb.na/servlet/f60servlet?config
=mult", LAST);
```

In the above example, the right boundary is `\r`. The actual right boundary may differ between systems.

Testing Oracle NCA Applications

The following sections contain several tips for testing secure Oracle NCA applications and servlets.

Testing Secure Oracle NCA Applications

- ▶ When selecting the protocols to record, you only need to select **Oracle NCA**—not **Web Protocol** from the protocol list. VuGen records the security information internally and therefore does not need the explicit Web functions.
- ▶ In the Port Mapping recording options, delete any existing entries for port 443 and create a new entry for the Oracle server name:

Service ID: HTTP

Target Server: Oracle Forms Server IP address or long host name

Target Port: 443

Connection Type: SSL

SSL Version: Active SSL version. If in doubt, select SSL 2/3.

For more information, see "Configuring the Port Mappings" in *Volume I-Using VuGen*.

- ▶ If you encounter problems when replaying an NCA HTTPS script during the `nca_connect_server` command, insert the following function at the beginning of the script.

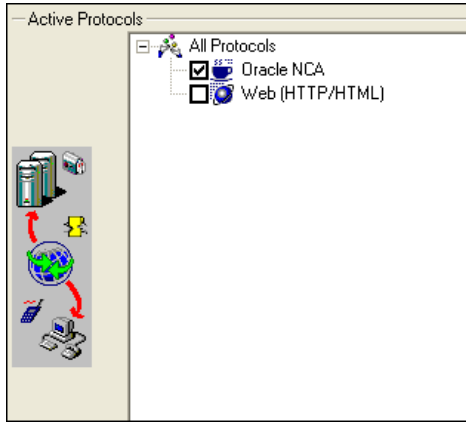
```
web_set_sockets_option("SSL_VERSION","3");
```

Testing Servlets and other Oracle NCA Applications

Certain NCA sessions use servlets:

- ▶ the Forms Listener servlet
- ▶ applications or modules that use both NCA and HTTP communications, such as the Oracle Configurator
- ▶ the initializing of the NCA application (downloading the applet, jar, and gif files)

When recording servlets, you must record both Oracle NCA and Web functions. You can do this by initially creating a multi-protocol script. Alternatively, if you created a single protocol script for Oracle NCA, open the **General:Protocols** node in the Recording Options, and enable the Web protocol. Then you can begin recording.



If you are unsure whether your application uses servlets, check the **default.cfg** file in the script directory. Locate the entry

UseServletMode=

If the value is 1 or 2, then servlets are being used and you must enable HTTP recording in addition to Oracle NCA.

If you already recorded a script, you can regenerate the code automatically to include the Web functions without having to re-record. Select **Tools > Regenerate Script**, and select the Web protocol in the Protocols section.

Determining the Recording Mode

When recording Oracle NCA scripts: VuGen automatically determines the correct connection mode: HTTP or Socket mode. Generally, you are not required to modify any of the recording settings as VuGen auto-detects the system configuration. In systems where the standard port mapping are reserved by other applications, you may need to modify the Port Mapping settings, depending on the recording mode.

You can determine the recording mode in one of the following ways:

- When using the NCA application, open the Java Console.

```
proxyHost=null
proxyPort=0
connectMode=HTTP
Forms Applet version is: 60812
```

The **connectMode** entry indicates **HTTP**, **HTTPS**, or **socket**.

- After recording an NCA session, open the **default.cfg** file in the **Vuser** directory and check the value of the **UseHttpConnectMode** entry.

```
[HttpConnectMode]
UseHttpConnectMode= 2
// 0 = socket 1 = http 2 = https
```

When defining a new port mapping 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 "Configuring the Port Mappings" in *Volume I-Using VuGen*.

Recording Trace Information for Oracle DB

To debug your script, you can use the Oracle DB breakdown graphs. To gather data for this graph, you turn on the trace mechanism before running the script.

To manually turn on the tracing mechanism, use the **nca_set_custom_dbtrace** function. For more information, see the *Online Function Reference* (**Help > Function Reference**).

Correlating Oracle NCA Statements for Load Balancing

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

To correlate statements for load balancing:

1 Record a multi-protocol script.

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

2 Define parameters for host and host arguments.

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

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

3 Assign values to `serverHost` and `serverArgs`:

```
web_url("step_name", "URL=http://server1.acme.com/test.htm", LAST);
```

4 Modify the `nca_connect_server` statement from:

```
nca_connect_server("199.203.78.170",
  9000/*version=107*/,
  "module=e:\appsna...fndnam=apps ");
```

to:

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

The script should now look like this:

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

Additional Recommended Correlations

When recording an Oracle NCA session, VuGen records dynamic values—values that change for each record and replay session. Two common dynamic arguments are `icx_ticket` and `JServSessionIdroot`.

icx_ticket

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

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

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

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

...

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

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

JServSessionIdroot

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

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

```
vuser_init.c(8): Set-Cookie: JServSessionIdroot=my1sanw2n1.JS4; path=/\r\n
vuser_init.c(8): Content-Length: 79\r\n
vuser_init.c(8): Content-Type: text/plain\r\n
vuser_init.c(8): \r\n
vuser_init.c(8): 81-byte response body for "http://ABC-
123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo&ifhost=mercury&ifip=12
3.45.789.12" (RelFrameId=1)
vuser_init.c(8):
/servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdroot=my1sanw2n1.JS4\r\n
```


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

```
web_reg_save_param("NCAJServSessionId","LB=\r\n\r\n","RB=\r","ORD=1",LAST)
;

web_url("f60servlet",
  "URL= http://ABC-"123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo&"
  "ifhost=mercury&ifip=123.45.789.12", LAST);

web_url("oracle.forms.servlet.ListenerSer",
  "URL=http://ABC-123{NCAJServSessionId}?ifcmd=getinfo&"
  "ifhost=mercury&ifip=123.45.789.12", LAST);
```

Recording in Pragma Mode

The client side of the Oracle NCA Vuser can be configured to send an additional header to the server named **Pragma**. The header is a counter that behaves in the following way: the initial message of the NCA handshake has a value of 1.

The messages that follow the handshake are counted, beginning with 3. The counter is incremented by 1 for each message sent by the client.

If the message received from the server is the type plain/text and the body of the message begins with ifError:##00, the client sends a 0 byte message to the server and the Pragma value changes its sign to a minus. This sign changes back after the client succeeds in receiving the information from the server.

Recording of the Pragma header is only supported in the multi-protocol mode (Oracle NCA and Web). You can identify the Pragma mode within the script's default.cfg file. When operating in Pragma mode, the UseServletMode is set to 2.

```
[HttpConnectMode]
UseHttpConnectMode=1
RelativeURL=<NCAJServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

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"
  ...
```


46

SAPGUI Protocol

In the growing field of ERP (Enterprise Resource Planning), SAP provides solutions allowing companies to manage all of their business processes. HP provides tools for testing SAP solution modules on both functional and load testing levels. This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). For information on testing solutions for mySAP Workplace and Portal clients, see Chapter 49, "SAP-Web Protocol."

This chapter includes:

- About Developing SAPGUI Vuser Scripts on page 694
- Checking your Environment for SAPGUI Vusers on page 695
- Creating a SAPGUI Vuser Script on page 706
- Recording a SAPGUI Vuser Script on page 707
- Inserting Steps Interactively into a SAPGUI Script on page 710
- Understanding a SAPGUI Vuser Script on page 712
- Enhancing a SAPGUI Vuser Script on page 716

About Developing SAPGUI Vuser Scripts

This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). To test the SAPGUI user operating only on the client, use the SAPGUI Vuser type. To test a SAPGUI user that also uses a Web browser, use the SAP (Click and Script) protocol.

Before recording a session, verify that your modules and client interfaces are supported by VuGen. The following table describes the SAP client modules for SAP Business applications and the relevant tools:

SAP module	VuGen support
SAP Web Client or mySAP.com.	Use the SAP-Web Vuser type.
SAPGUI for Windows.	A Windows-based client, emulated by the SAPGUI Vuser. This also supports APO module recording (requires patch level 24 for APO 3.0).
SAPGUI for Windows and a web browser.	Use the SAP (Click and Script) protocol.
SAPGUI for Java.	This client is not supported.

Version 6.20 and later:

- ▶ **For Functional Testing.** Use the QuickTest Professional Add-in for mySAP.com client.
- ▶ **For Load Testing.** Use the SAPGUI or SAP (Click and Script) protocol to create a script in VuGen and run a scenario in the Controller.

You use VuGen to record typical business processes. VuGen records SAPGUI for Windows client activity during SAP business processes, and generates a Vuser script. When you perform actions within the SAPGUI for Windows client, VuGen generates functions that describe this activity. Each function begins with a **sapgui** prefix.

Checking your Environment for SAPGUI Vusers

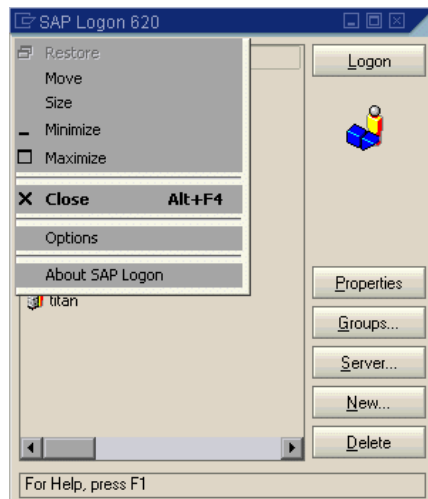
The basic steps in checking and setting up your system for the recording of SAPGUI Vusers, are Checking the Patch Level and Enabling Scripting. Once your environment is configured properly, you can record a typical SAP session and replay it in VuGen.

Checking the Patch Level

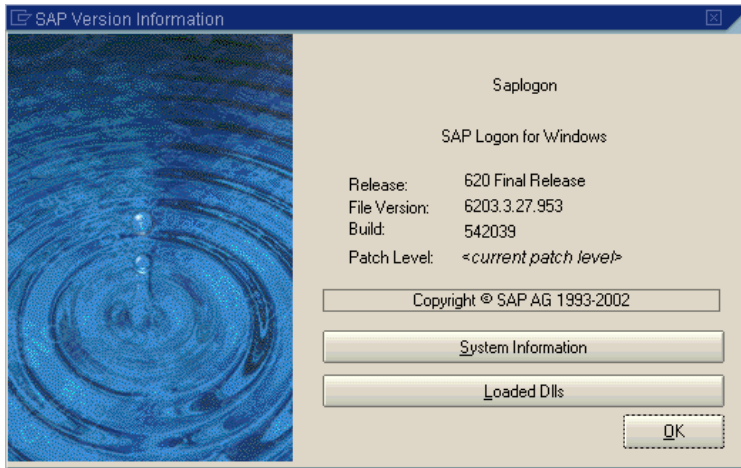
You can check the patch level of your SAPGUI for Windows client from the About box. The lowest patch level supported is version 6.20 patch 32.

To check the patch level:

- 1 Invoke the SAPGUI logon window. Click the top left corner of the SAP Logon dialog box and select **About SAP Logon** from the menu.



- 2 The SAP version information dialog box opens. Verify that the Patch Level entry is 32 or higher.



Enabling Scripting

VuGen support for the SAPGUI for Windows client, is based on SAP's Scripting API. This API allows Vusers to interact with the SAPGUI client, receive notifications, and perform operations.

The Scripting API is only available in recent versions of the SAP Kernel. In kernel versions that support scripting, the option is disabled by default. In order to use VuGen, first make sure that the SAP servers support the Scripting API, and enable the API on both the server and clients. For more information and to download patches, see the SAP OSS note #480149.

VuGen provides a utility that checks if your system supports scripting. The utility, **VerifyScript.exe**, is located on DVD in the **Additional Components\SAP_Tools\VerifySAPGUI** folder. For more information, see the file **VerifyScripting.htm** provided with this utility.

The following sections describe how to enable scripting.

- Checking the Configuration
- Enabling Scripting on the SAP Application Server
- Enabling Scripting on SAPGUI 6.20 Client

Checking the Configuration

The first step in enabling scripting is ensuring that the right kernel version is installed, and updating it if required.

Check the table below, for the minimum kernel patch level required for your version of the SAP Application Server. If required, download and install the latest patch.

Software Component	Release	Package Name	Kernel Patch Level
SAP_APPL	31I	SAPKH31I96	Kernel 3.1I level 650
SAP_APPL	40B	SAPKH40B71	Kernel 4.0B level 903
SAP_APPL	45B	SAPKH45B49	Kernel 4.5B level 753
SAP_BASIS	46B	SAPKB46B37	Kernel 4.6D level 948
SAP_BASIS	46C	SAPKB46C29	Kernel 4.6D level 948
SAP_BASIS	46D	SAPKB46D17	Kernel 4.6D level 948
SAP_BASIS	610	SAPKB61012	Kernel 6.10 level 360

To check the kernel patch level:

- 1 Log in to the SAP system
- 2 Select **System > Status**
- 3 Click the **Other kernel information button** (with the yellow arrow).



System: Status

Usage data

Client	800	Previous logon	17.10.2002	13:53:52
User	SUPER	Logon		13:55:11
Language	EN	System time		13:55:15

SAP data

Repository data

Transaction	SESSION_MANA...
Program (screen)	SAPLSMTR_NAV...
Screen number	100
Program (GUI)	SAPLSMTR_...
GUI status	SESSION_ADMIN

SAP System data


Component version	R/3 Release 4...
Installation number	0120033759
License expiry date	31.12.9999

Host data

Operating system	Windows NT
Machine type	2x Intel 8
Server name	calderone_MI...
Platform ID	560

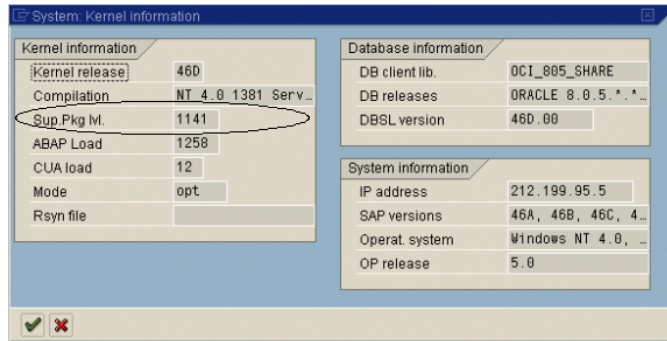
Database data

System	ORACLE
Release	8.1.7.0.0
Name	MI6
Host	CALDERONE
Owner	SAPR3

✓ Navigate  ✗

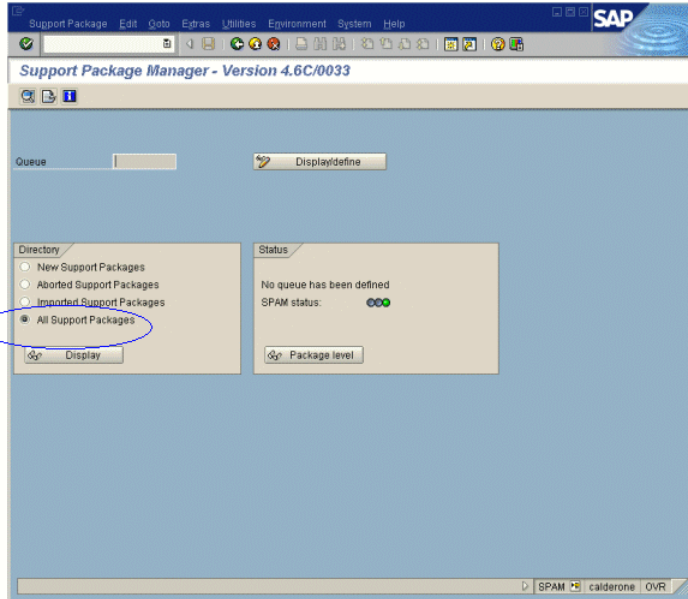
4 In the **Kernel Information** section, check the value of the **Sup. Pkg. lvl.**

If the level is lower than 948, you must download the latest kernel version and upgrade your existing one. See the SAP OSS note #480149 for detailed instructions on how to perform this upgrade.

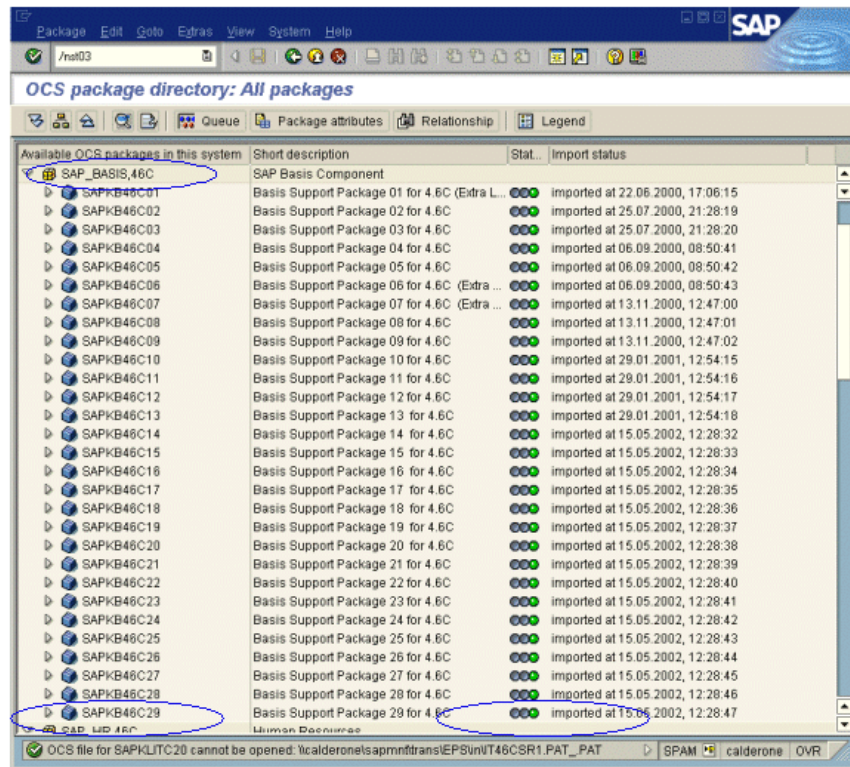


To check the R/3 support packages:

- 1** Log on to the SAP system and run the SPAM transaction.
- 2** In the **Directory** section, select **All Support Packages**, and click the **Display** button.



- Verify that SAPKB46C29 is installed for SAP_BASIS, 4.6C. If it is installed, a green circle appears in the Status column.



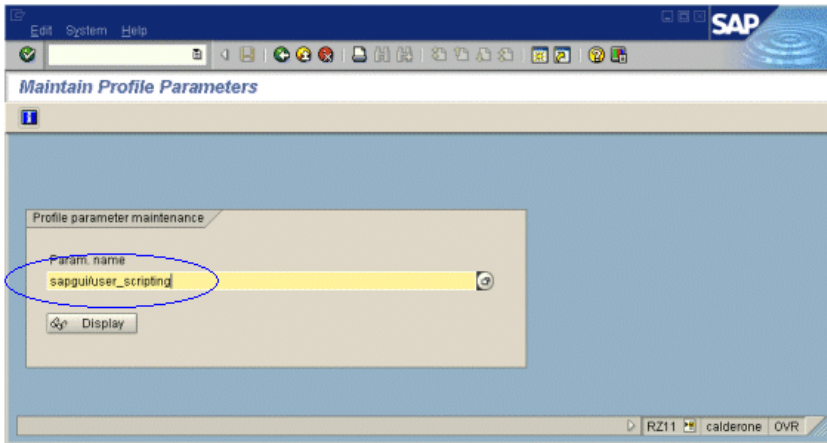
If you do not have the OCS package installed, download it from the www.sap.com Web site and install it. For more information, see the SAP OSS note #480149.

Enabling Scripting on the SAP Application Server

A user with administrative permissions enables scripting by setting the **sapgui/user_scripting** profile parameter to **TRUE** on the application server. To enable scripting for all users, set this parameter on all application servers. To enable scripting for a specific group of users, only set the parameter on application servers with the desired access restrictions.

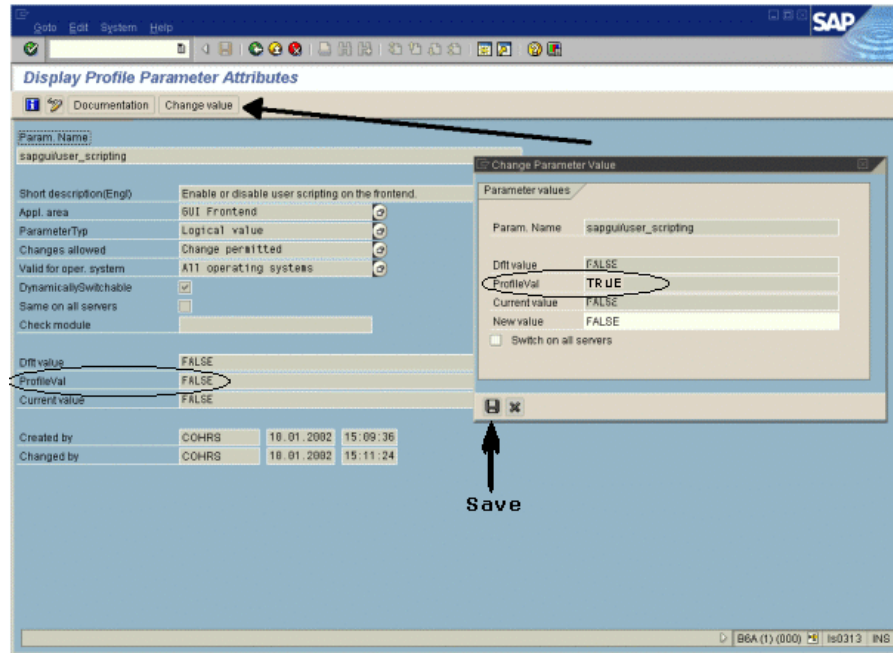
To change the profile parameter:

- 1 Open transaction **rz11**. Specify the parameter name **sapgui/user_scripting** and click **Display**. The Display Profile Parameter Attributes window opens.



If **Parameter name is unknown** appears in the status bar, this indicates that you are missing the current Support Package. Import the Support Package that corresponds to the SAP BASIS and kernel versions of the application server, as described in "Checking the Configuration" on page 697.

- 2 If **Profile Val** is FALSE, you need to modify its value. Click the **Change value** button in the toolbar. The Change Parameter Value window opens. Enter TRUE in the **ProfileVal** box and click the **Save** button.



When you save the change, the window closes and **ProfileVal** is set to TRUE.

- 3 Restart the application server, since this change only takes effect when you log onto the system.

If the updated **ProfileVal** did not change, even after restarting the server, then the kernel of the application server is outdated. Import the required kernel patch, as specified in the section "Checking the Configuration" on page 697.

Note that the Profile Value may be dynamically activated in the following kernel versions, using transaction rz11, without having to restart the application server.

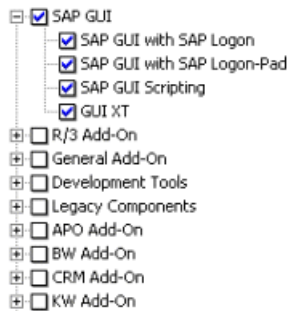
Release	Kernel Version	Patch Level
4.6B, 4.6C, 4.6D	4.6D	972
6.10	6.10	391
6.20	all versions	all levels

Enabling Scripting on SAPGUI 6.20 Client

To allow VuGen to run scripts, you must also enable scripting on the SAPGUI client. You should also configure the client not to display certain messages, such as when a connection is established, or when a script is attached to the GUI process.

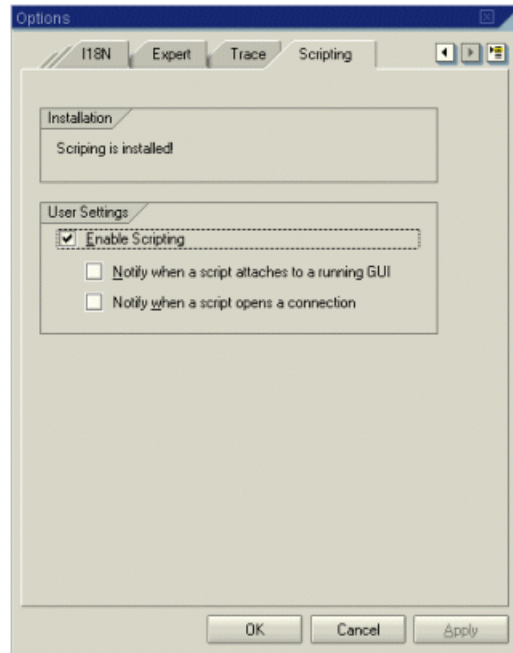
To configure the SAPGUI client to work with VuGen:

- ▶ **During installation.** While installing the SAPGUI client, enable the **SAP GUI Scripting** option.



► **After installation.** Suppress warning messages. Open the Options dialog box in the SAPGUI client. Select the **Scripting** tab and clear the following options:

- 1 Notify when a script attaches to a running GUI**
- 2 Notify when a script opens a connection**



You can also prevent these messages from popping up by setting the values **WarnOnAttach** and **WarnOnConnection** in the following registry key to 0:

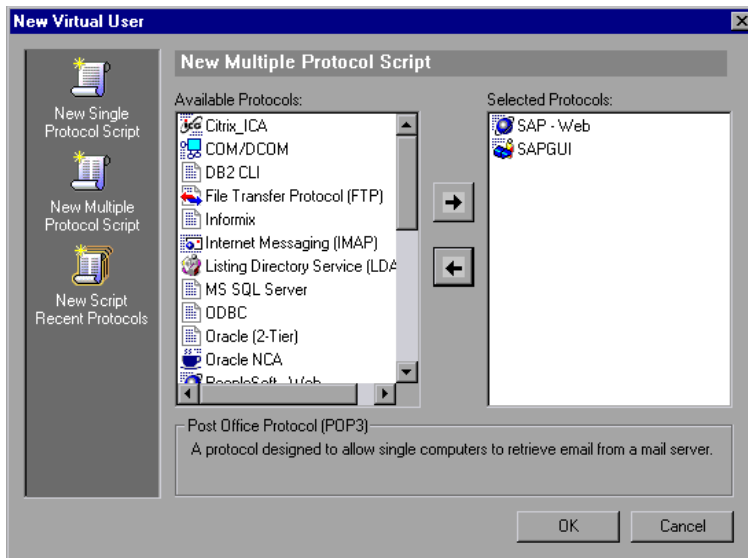
HKCU\SOFTWARE\SAP\SAPGUI Front\SAP Frontend Server\Security.

Creating a SAPGUI Vuser Script

The first step in creating a SAPGUI Vuser script is choosing the Vuser and script type. The SAP Vuser type, **SAPGUI** is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script.

To create a SAPGUI Vuser script:

- 1** Invoke VuGen and select **File > New**.
- 2** To record a standard SAPGUI client session (with no browser controls), create a single-protocol Vuser script using the **SAPGUI** type Vuser.
- 3** To record a SAPGUI session that uses browser controls, create a multi-protocol Vuser script. Specify both the **SAPGUI** and **SAP-Web** Vuser types. This allows VuGen to record Web-specific functions when encountering the browser controls.



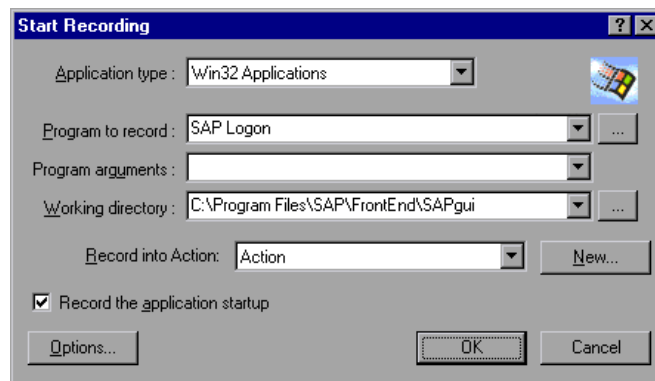
- 4** Click **OK** to open the Vuser script.

Recording a SAPGUI Vuser Script

After creating an empty script, you set the recording options and then record your SAPGUI session. VuGen generates a script corresponding to your actions within the client.

To begin recording a SAPGUI script:

- 1 If the Start Recording dialog box was not opened, click the **Start Recording** button. The Start Recording dialog box opens.
- 2 VuGen detects and fills in the relevant information:



- **Program to record.** VuGen locates the saplogon.exe file in the SAP client installation.
 - **Working Directory.** For applications that require you to specify a working directory, specify it here. The required information differs, depending on the type of Vuser script.
 - **Record into Action.** Select the section into which you want to record. Initially, the available sections are vuser_init, Action1, and vuser_end
- 3 Click **OK** and begin recording.

Recording at the Cursor

VuGen also allows you to record actions into an existing script. You may decide to record into an existing script for several reasons:

- ▶ You made a mistake in the actions that you performed during recording.
- ▶ Your actions were correct, but you need to add additional information such as the handling of popup windows. For example the SAP server may issue an inventory warning, which did not apply during the recording session.

This feature, called Recording at the Cursor, lets you insert new actions or replace existing actions. When you begin Recording at the Cursor, VuGen prompts you with two options:

- ▶ **Insert steps into action.** Inserts the newly recorded steps at the cursor without overwriting any existing steps. The new segment is enclosed with comments indicating the beginning and end of the added section. This option is ideal for handling occasional popup windows that were not present during the recording

```
// Recording at the cursor - Begin
  sapgui_select_active_connection("con[0]");
  sapgui_select_active_session("ses[0]");
  sapgui_select_active_window("wnd[0]");
//Recording at the cursor - End
```

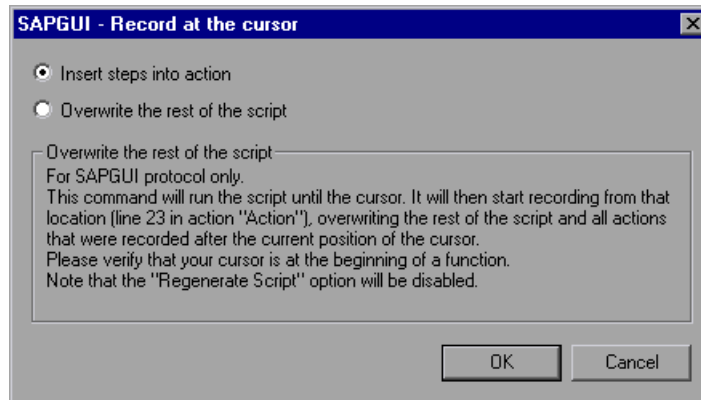
- ▶ **Overwrite the rest of the script.** Replaces all steps from the point of the cursor onward. This option overwrites the remainder of the current Action and deletes all other Actions. It does not affect the **vuser_init** or **vuser_end** sections.

After you select one of the Recording at the Cursor options, VuGen replays the script from the beginning until the cursor location. Then it opens the Recording floating toolbar and begins recording. If you use the **Recording at the Cursor** feature, the **Regenerate Script** tool becomes disabled.

Note: To record at the cursor, you need to click in the left margin of the VuGen editor, immediately before an existing function.

To Record at the Cursor:

- 1 Open Script view (**View > Script view**) and click in the left margin adjacent to an existing function.
- 2 Click the **Recording at the Cursor** button. VuGen prompts you to make a selection.



- 3 Select **Insert steps into action** or **Overwrite the rest of the script**. Click **OK**. VuGen replays the script until the point of the cursor.
- 4 Wait for the Recording floating toolbar to open. Then begin performing actions in the SAPGUI client, switching between sections and actions as required.



- 5 Click the **Stop** button to end the recording session.

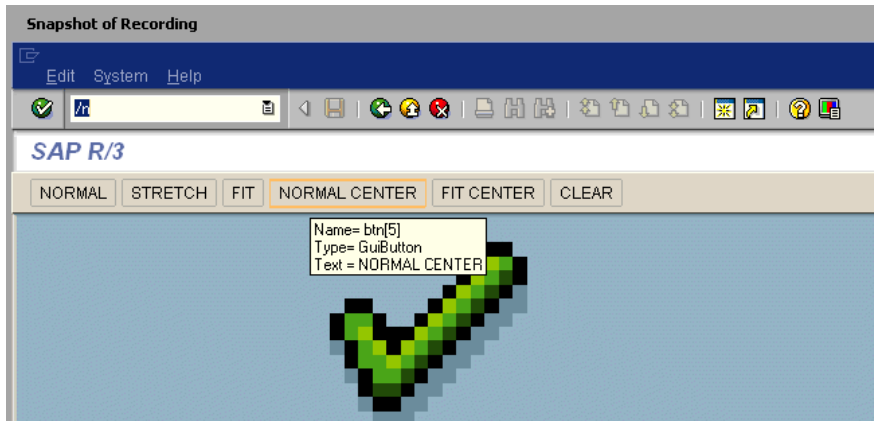
Inserting Steps Interactively into a SAPGUI Script

After recording, you can manually add steps to the script in either Script view and Tree View. For information about adding steps from the various views, see "Introducing Service TestVuGen" in *Volume I-Using VuGen*.

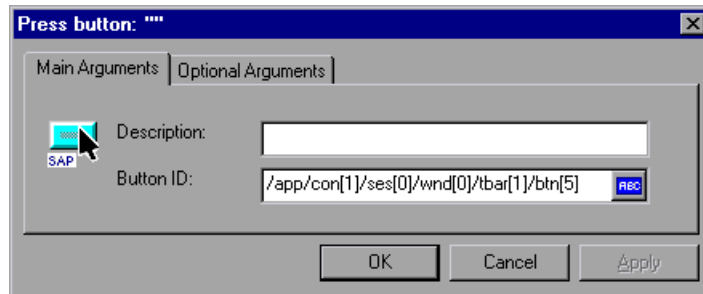
In addition to manually adding new functions, you can add new steps interactively for SAPGUI Vusers, directly from the snapshot. Using the right-click menu, you can add object-related steps.

When adding a step from within a snapshot, VuGen uses the Active Screen capability and determines the ID of each object in the SAPGUI client window (unless you disabled Active Screen snapshots in the SAPGUI General Recording Options).

To determine which objects were recognized by VuGen, you move the mouse over the snapshot. VuGen draws a box around the objects as you pass over them and displays a tool tip with the object's Control ID. In the following example, the selected active object is the NORMAL CENTER button.



When you add a step while holding the mouse over a recognized object, VuGen automatically inserts the Control ID of that object into the relevant field of the Properties dialog box. For example, if you add a **Press Button** step, for the NORMAL CENTER button as shown above, the Properties box displays the following ID:



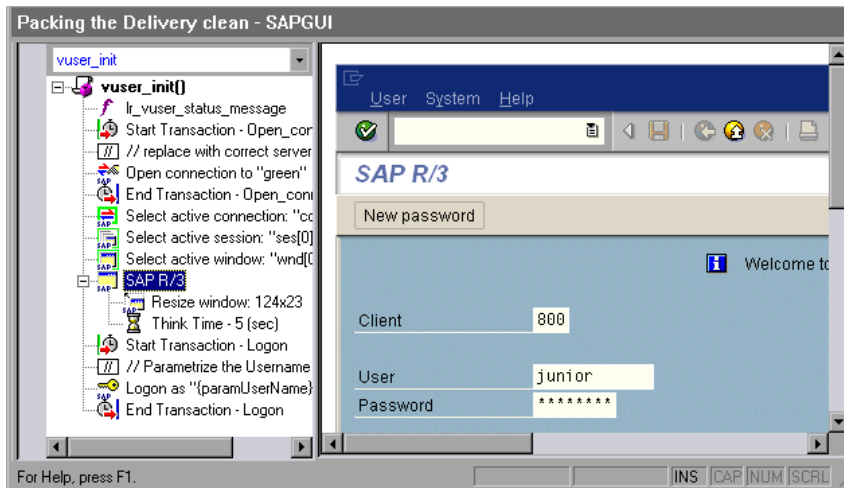
To insert a step interactively for a specific object:

- 1 Click within the Snapshot window.
- 2 Move the mouse over the object for which you want to add a function. Make sure that VuGen recognizes the object and encloses it with a box.
- 3 Select **Insert New Step** from the right-click menu. The Insert Step box opens.
- 4 Select a step from the menu. The step's Properties dialog box opens, with the Control ID of the object when relevant.
- 5 Enter a name for the object in the **Description** box. Click **OK**. VuGen inserts the new step after the selected step.
- 6 To get the Control ID of the object for the purpose of pasting it into a specific location, select **Copy Control ID** from the right-click menu. VuGen places it on the clipboard. You can paste it into a Properties box or directly into the code from the Script view.

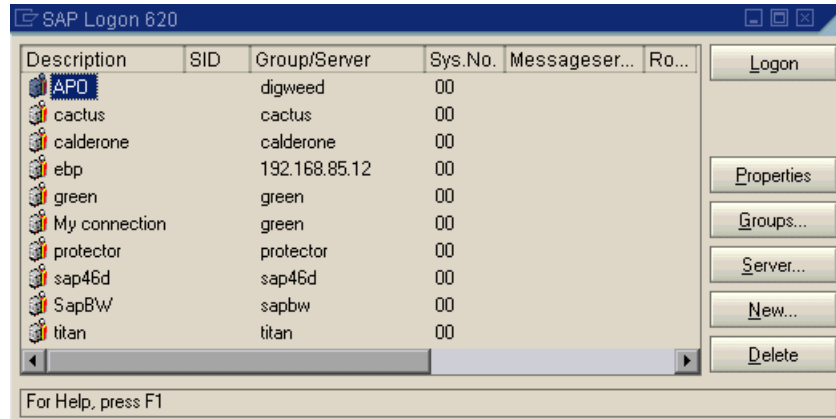
Understanding a SAPGUI Vuser Script

The SAPGUI Vuser script typically contains several SAP transactions which make up a business process. A business process consists of functions that emulate user actions. Open the tree view to see each user action as a Vuser script step.

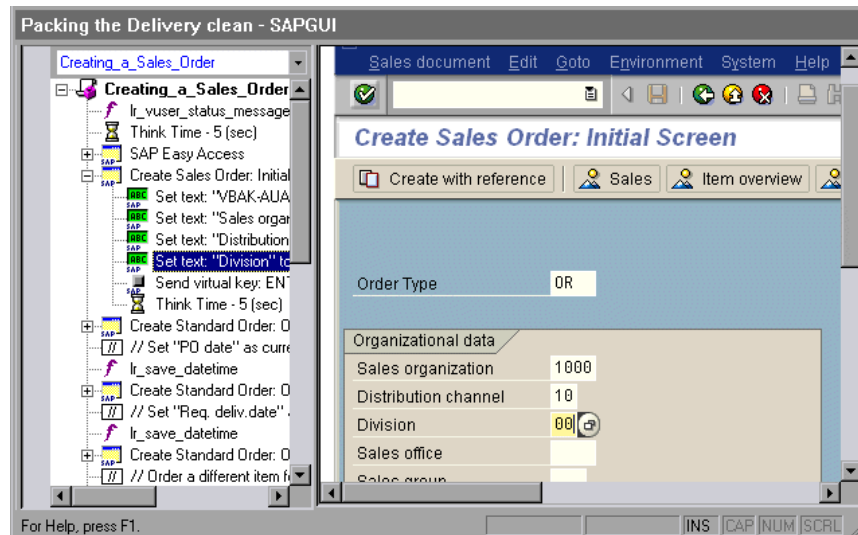
The following example shows a typical recording of a SAPGUI client. The first section, **vuser_init**, contains the opening of a connection and a logon.



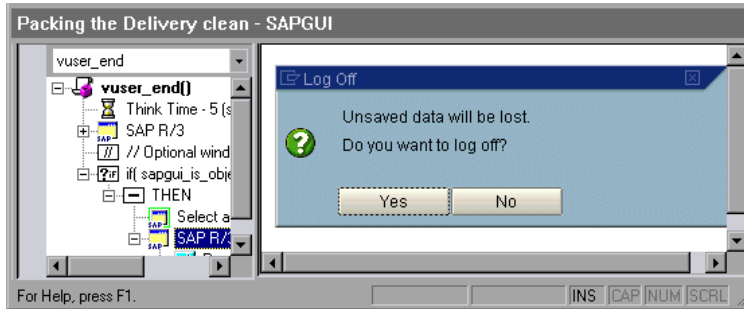
Note that the Open Connection step uses one of the connection names in the SAP Logon **Descriptions** list. If the specified connection name is not in the list, the Vuser looks for a server with that name.



In the following section, the functions emulate typical user operations such as menu selection and the setting of a check box.



The final section, **vuser_end**, illustrates the logoff procedure.



When recording a multi- protocol script for both SAPGUI and Web, VuGen generates steps for both protocols. In the Script view, you can view both **sapgui** and **web** functions.

The following example illustrates a multi-protocol recording in which the SAPGUI client opens a Web control. Note the switch from **sapgui** to **web** functions.

```
sapgui_tree_double_click_item("Use as general WWW browser, REPTITLE",
    "shellcont/shell",
    "000732",
    "REPTITLE",
    BEGIN_OPTIONAL,
    "AdditionalInfo=sapgui1020",
    END_OPTIONAL);

...

sapgui_set_text("",
    "http:\\\\yahoo.com",
    "usr/txtEDURL",
    BEGIN_OPTIONAL,
    "AdditionalInfo=sapgui1021",
    END_OPTIONAL);

...

web_add_cookie("B=7pt5cisv1p3m2&b=2; DOMAIN=www.yahoo.com");

web_url("yahoo.com",
    "URL=http://yahoo.com/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    EXTRARES,
    "URL=http://srd.yahoo.com/hpt1/ni=17/ct=lan/sss=1043752588/t1=1043752575385/d1
=1251/d2=1312/d3=1642/d4=4757/0.4097009487287739/*1",
    "Referer=http://www.yahoo.com/", ENDITEM,
    LAST);
```

Enhancing a SAPGUI Vuser Script

After you examine the recorded Vuser script, you enhance it in the following ways:

- ▶ **Transactions.** Inserting transactions, rendezvous points, and control-flow structures into the script. For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.
- ▶ **Verification.** Insert SAPGUI verification functions to verify the current state of SAPGUI objects. For details, see Adding Verification Functions.
- ▶ **Retrieve information.** Insert SAPGUI functions to verify the current value of SAPGUI objects. You use the `sapgui_get_xxx` functions to retrieve information. For more information, see "Retrieving Information" on page 717.

Define parameters (optional). Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values. For details, see "Creating Parameters" in *Volume I-Using VuGen*.

Adding Verification Functions

When working with optional or dynamic windows or frames, you can use verification functions to determine if the window or object is available. An optional window is a window that does not consistently open during the SAP session. This function allow the Vuser script to continue running even if an optional window opens or an exception occurs.

The first example checks if a window is available. If the window is available, the Vuser closes it before continuing.

```
if (!sapgui_is_object_available("wnd[1]"))
    sapgui_call_method("{ButtonID}",
        "press",
        LAST,
        AdditionalInfo=info1011");
sapgui_press_button(.....)
```

The next example illustrates a dynamic object in the ME51N transaction. The Document overview frame is optional, and can be opened/closed by the **Document overview on/off** button.

The code checks the text on the Document overview button. If the text on the button shows Document overview on, we click the button to close the Document overview frame.

```

if(sapgui_is_object_available("tbar[1]/btn[9]"))
{
    sapgui_get_text("Document overview on/off button",
        "tbar[1]/btn[9]",
        "paramButtonText",
        LAST);

    if(0 == strcmp("Document overview off", lr_eval_string("{paramButtonText}")))
        sapgui_press_button("Document overview off",
            "tbar[1]/btn[9]",
            BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui1013",
            END_OPTIONAL);
}

```

Retrieving Information

When working with SAGUI Vusers, you can retrieve the current value of a SAPGUI object using the `sapgui_get_<xxx>` functions. You can use this value as input for another business process, or display it in the output log.

Retrieving Status Bar Information

The following example illustrates how to save part of a status bar message in order to retrieve the order number.

To retrieve the order number from the status bar:

- 1** Navigate to the point where you want to check the status bar text, and select **Insert > New Step**. Select the `sapgui_status_bar_get_type` function. This verifies that the Vuser can successfully retrieve text from the status bar.
- 2** Insert an **if** statement that checks if the previous statement succeeded. If so, save the value of the argument using `sapgui_status_bar_get_param`.

This `sapgui_status_bar_get_param` function saves the order number into a user-defined parameter. In this case, the order number is the second index of the status bar string.

```
sapgui_press_button("Save (Ctrl+S)",
  "tbar[0]/btn[11]",
  BEGIN_OPTIONAL,
  "AdditionalInfo=sapgui1038",
  END_OPTIONAL);

sapgui_status_bar_get_type("Status");
if(0==strcmp(lr_eval_string("{Status}"),"Success"))
  sapgui_status_bar_get_param("2", " Order_Number ");
```

During test execution, the Execution log indicates the value and parameter name:

```
Action.c(240): Pressed button " Save (Ctrl+S)"
Action.c(248): The type of the status bar is "Success"
Action.c(251): The value of parameter 2 in the status bar is "33232"
```

Saving Date Information

When creating scripts that use dates, your script may not run properly. For example, if you record the script on June 2, and replay it on June 3, the date fields will be incorrect. Therefore, you need to save the date to a parameter during text execution, and use the stored value as input for other date fields. To save the current date or time during script execution, use the **`lr_save_datetime`** function. Insert this function before the function requiring the date information. Note that the format of the date is specific to your locale. Use the relevant format within the **`lr_save_datetime`** function. For example, for month.day.year, specify "%m.%d.%Y".

In the following example, `lr_save_datetime` saves the current date. The `sapgui_set_text` function uses this value to set the delivery date for two days later.

```
lr_save_datetime("%d.%m.%Y", DATE_NOW + (2 * ONE_DAY),
  "paramDateTodayPlus2");

sapgui_set_text("Req. deliv.date",
  "{paramDateTodayPlus2}",
  "usr/ctxtRV45A-KETDAT",
  BEGIN_OPTIONAL,
  "AdditionalInfo=sapgui1025",
  END_OPTIONAL);
```


47

SAPGUI - Replaying Scripts

After creating a SAPGUI script through recording and manual enhancements, you replay it in VuGen to test its functionality.

This chapter includes:

- ▶ About Replaying SAPGUI Vuser Scripts on page 721
- ▶ Replaying SAPGUI Optional Windows on page 722
- ▶ SAPGUI Functions on page 723
- ▶ Tips for SAPGUI Vuser Scripts on page 724
- ▶ Troubleshooting SAPGUI Vuser Scripts on page 728
- ▶ Additional Resources on page 730

The following information applies to the SAPGUI and the SAP (Click and Script) protocols.

About Replaying SAPGUI Vuser Scripts

This chapter discusses the running of a SAPGUI Vuser script. You can set run-time settings to control the Vuser's behavior during the test or monitoring session.

This chapter also contains several guidelines for working with SAPGUI Vusers, as well as a troubleshooting section for solving common issues.

The SAPGUI Vuser script emulates a typical business processes using SAPGUI functions that begin with a **sapgui** prefix.

During replay, these functions emulate user activity on SAPGUI objects.

For example, `sapgui_select_radio_button` selects the radio button Blue.

```
sapgui_select_radio_button("Blue",  
    "usr/radRB7",  
    BEGIN_OPTIONAL,  
    "AdditionalInfo=sapgui1027",  
    END_OPTIONAL);
```

Replaying SAPGUI Optional Windows

When working with SAPGUI Vuser Scripts, you may encounter optional windows in the SAPGUI client—windows that were present during recording, but do not exist during replay. If you try to replay your recorded script as is, it will fail when it attempts to find the missing windows.

VuGen's optional window mechanism performs the actions on a window only after verifying that it exists. The Vuser checks if the window indicated in the **Select active window** step exists. If the window is found during replay, it performs the actions as they were recorded in the script. If it does not exist, the Vuser ignores all window actions until the next **Select active window** step. Note that only SAPGUI steps (beginning with a **sapgui** prefix) are ignored.

To use this feature, in Tree view select the appropriate Select Active Window step and select **Run steps for window only if it exists** from the right-click menu.

To disable this feature and attempt to run these steps at all times, regardless of whether the Vuser finds the window or not, select **Always run steps for this window** from the right-click menu.

SAPGUI Functions

During a SAPGUI recording session, VuGen generates functions that emulate user interaction with the SAPGUI client. When you record the SAPGUI for Windows client, VuGen generates functions with a **sapgui** prefix. This section lists all of the **sapgui** functions.

When you record a SAP session using a Web interface such as SAP Workplace or Portal, or if the SAPGUI client opens a Web control, VuGen generates functions with a **web** prefix.

While most of the functions are recorded, you can manually insert any function into your script. The functions that are not recorded are the data retrieval functions beginning with **sapgui_get**, and those used for verification, beginning with **sapgui_is**.

There are several categories of **sapgui** functions: Connection and Session Functions, Method and Property Functions, Verification and Data Retrieval Functions, and Object functions. Object functions are those which perform an action within a SAPGUI object such as Calendar Functions, Grid Functions, APO Grid Functions, Status Bar Functions, Table Functions, Tree Functions, Window Functions, and General Object Functions.

For more information about the **sapgui** and **web** functions, use the **Show Function Syntax** feature from the Edit menu, or see the *Online Function Reference* (**Help > Function Reference**).

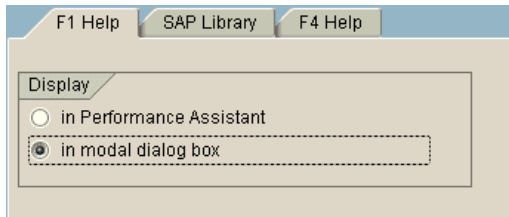
Tips for SAPGUI Vuser Scripts

The following sections provides Recording Tips, Replay Tips, and Tips for Replaying in a Scenario for SAPGUI Vusers. In addition, you can obtain information directly from the SAP support site.

Recording Tips

This section provides recording tips for a SAPGUI Vuser script.

- ▶ Make sure to record the actions into the appropriate sections: Record the logon procedure into the **vuser_init** section, the actions that you want to repeat in the **Actions** sections, and the logoff procedure in the **vuser_end** section.
- ▶ When recording a multi-protocol script in which the SAPGUI client contains Web controls, close the SAPLogon application before recording.
- ▶ Use modal dialog boxes for F1. Instruct the SAPGUI client to open the F1 help in a modal dialog box. Select **Help >Settings**. Click the **F1 Help** tab and select the **in modal dialog box** option in the Display section.

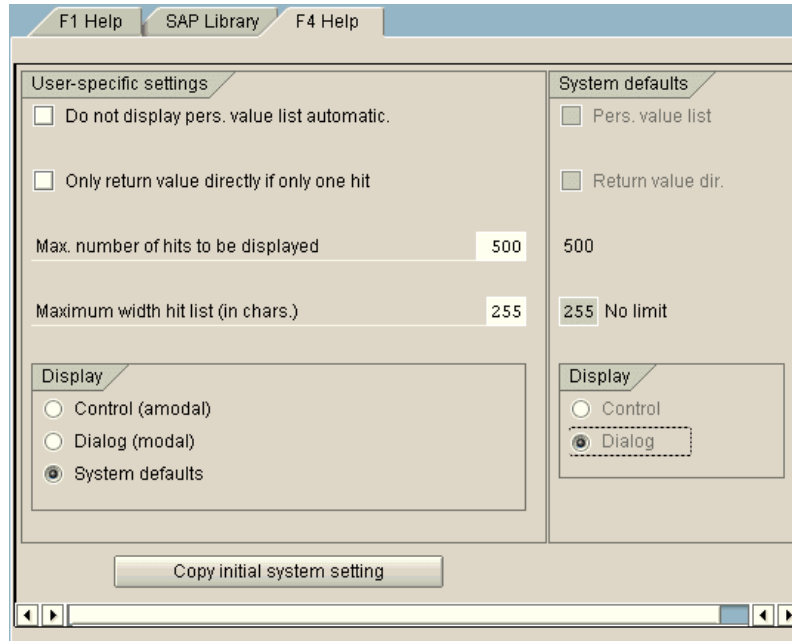


- ▶ Use modal dialog boxes for F4. Instruct the SAPGUI client to open the F4 help in a modal dialog box.

The following procedure must be performed by a SAP administrator:

To open F4 help in modal dialog boxes:

- 1** Make sure that all users have logged off from the server.
- 2** Select **Help > Settings**. Click the **F4 Help** tab.



- 3** In the Display section (bottom left), select System defaults.
- 4** In the Display portion of the System defaults section (bottom right), select **Dialog**.
- 5** Save the changes—click **Copy initial system setting** or CTRL+S.
- 6** Verify that the status bar displays the message **Data was saved**.
- 7** Close the session.
- 8** Restart the service through the SAP Management Console.

Replay Tips

Follow these guidelines before replaying your script in standalone-mode:

- ▶ Replace the encrypted password in the **sapgui_logon** function generated during recording, with the real password. It is the second argument of the function, after the user name: `sapgui_logon("user", "pswd", "800", "EN");`
For additional security, you can encrypt the password within the code. Select the password text (the actual text, not *****) and select **Encrypt string** from the right-click menu. VuGen inserts an **lr_decrypt** function at the location of the password: `sapgui_logon("user", lr_decrypt("3ea037b758"), "800", "EN");`.
- ▶ When running a script for the first time, configure VuGen to show the SAPGUI user interface during replay, in order to see the operations being performed through the UI. To show the user interface during replay, open the run-time settings (F4) and select the **Show SAP Client During Replay** option in the **SAPGUI:General** node. During a load scenario, disable this option, since it uses a large amount of system resources in displaying the UI for multiple Vusers.

Tips for Replaying in a Scenario

The following sections provide configuration tips for running the script on a Controller or Load Generator machine.

Controller Settings

When working with a LoadRunner scenario, set the following values when running your script in a load test configuration:

- ▶ **Ramp-up.** One by one (to insure proper logon) in the Scheduler.
- ▶ **Think time.** Random think time in the Run-Time settings.
- ▶ **Users per load generator.** 50 Vusers for machine with 512 MB of memory in the Load Generators dialog box.

Load Generator Settings

When running your script in a scenario, check the agent mode and configure the terminal sessions on the Load Generator machines.

- **Agent Mode.** Make sure that the LoadRunner (or Performance Center) Remote Agent is running in Process mode. Service mode is not supported.

To check this, move your mouse over the agent's icon in the Windows task bar area, and read the description. If the description reads LoadRunner Agent Service, it is running as a service.



To restart the agent as a process:



- 1** Stop the agent. Right-click the LoadRunner Agent icon and select **Close**.
- 2** Run **magentproc.exe**, located in the **launch_service\bin** directory, under the LoadRunner installation.
- 3** To make sure that the correct Agent is launched the next time you start your machine, change the Start type of the Agent Service from Automatic to Manual. Then add a shortcut to **magentproc.exe** to the Windows Startup folder.

Terminal Sessions. Machines running SAPGUI Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine. To increase the number of Vusers per machine, open additional terminal server sessions on the Load Generator machines. Select **Agent Configuration** from **Start > Programs > <product_name> > Advanced Settings**, and select the **Enable Terminal Service** option. You specify the number of terminal sessions in the Load generator machine properties. For more information, see *Configuring Terminal Services* in the *HP LoadRunner Controller User's Guide*.

Note: When the LoadRunner Agent is running in a terminal session, and the terminal session's window is minimized, no snapshots will be captured on errors.

Troubleshooting SAPGUI Vuser Scripts

Question 1: I was able to record a script, but why does replay fail?

Answer: In LoadRunner, make sure that the LoadRunner Remote Agent is running in Process mode. Service mode is not supported. For more information, see "Replay Tips" on page 726.

Question 2: Why were certain SAPGUI controls not recorded?

Answer: Some SAPGUI controls are only supported in their menu or toolbar contexts. Try performing the problematic task using a different means—through a menu option, context menu, toolbar, and so on.

Question 3: Why can't I record or replay any scripts in VuGen?

Answer:

- a** Verify that you have the latest patch of SAPGUI 6.20 installed. The lowest allowed patch level is patch 32.
- b** Make sure that scripting is enabled. See the "Checking your Environment for SAPGUI Vusers" on page 695.
- c** Verify that notifications are disabled in the SAPGUI for Windows client. Click the Customizing of Local Layout button or press ALT+F12. Click **Options** and select the Scripting tab. Clear both **Notify** options.

Question 4: What is the meaning of the error popup messages that are issued when I try to run the script?

Answer: Certain SAP applications store the last layout for each user (such as which frames are visible or hidden). If the stored layout was changed since the script was recorded, this may cause replay problems. For Example, in the ME52N transaction, the **Document overview Off/On** button will change the number of visible frames.

If this occurs:

- 1** Navigate the transaction to the same point as it was during recording, before starting replay. You can use the Snapshot viewer to see the layout in which it was recorded.
- 2** Add statements to the script that bring the transaction to the desired layout during replay. For example, if an optional frame interferes with your replay, insert a verification function that checks if the frame is open. If it is open, click a button to close it. For verification examples, see "Adding Verification Functions" on page 716.

Question 5: Can I use the single sign-on mechanism when running a script on a remote machine?

Answer: No, VuGen does not support the single sign-on connection mechanism. In your SAPGUI client, open the Advanced Options and clear the **Enable Secure Network Communication** feature. Note that you must re-record the script after you modify the Connection preferences.

Question 6: Can VuGen record all SAP objects?

Answer: Recording is not available for objects not supported by SAPGUI Scripting. See your recording log for information about those objects.

Question 7: Are all business processes supported?

Answer: VuGen does not support business processes that invoke Microsoft Office module controls, nor those that require the use of GuiXT. You can disable **GuiXT** from the SAPGUI for Windows client Options menu.

Additional Resources

LoadRunner

For Online Help on dialog boxes, press F1 within a dialog box. You can also select **Help > Contents and Index** to manually open the Help. In the Index tab, locate the **SAPGUI Vuser scripts** entry and click the appropriate sub-entry.

For Online Help with a function, click within the function or step, and click F1 to open the *Online Function Reference*.

SAP

For more information, see the SAP website at www.sap.com or one of the following locations:

- ▶ **SAP Notes** - <https://websmp103.sap-ag.de/notes>

Note #480149: New profile parameter for user scripting on the front end

Note #587202: Drag & Drop is a known limitation of the SAPGUI interface

- ▶ **SAP Patches** - <https://websmp104.sap-ag.de/patches>

SAP GUI for Windows - SAPGUI 6.20 Patch (the lowest allowed level is 32)

48

SAP (Click and Script) Protocol

VuGen allows you to create scripts that emulate SAP applications over the Web.

This chapter includes:

- ▶ About Developing SAP (Click and Script) Vuser Scripts on page 731
- ▶ Recording a SAP (Click and Script) Session on page 732
- ▶ Understanding SAP (Click and Script) Scripts on page 732

About Developing SAP (Click and Script) Vuser Scripts

VuGen can create test scripts for SAP Enterprise portal7 and SAP ITS 6.20/6.40 environments using specialized test objects and methods that have been customized for SAP. The objects are APIs based on HP QuickTest support for SAP.

As you record a test or component on your SAP application, VuGen records the operations you perform. VuGen recognizes special SAP Windows objects such as frames, table controls, iViews, and portals.

VuGen supports recording for the following SAP controls: button, checkbox, drop-down menu, edit field, iview, list, menu, navigation bar, OK code, portal, radio group, status bar, tab strip, table, and tree view.

Recording a SAP (Click and Script) Session

To create a Vuser script that emulates SAP Web applications, you select the **SAP (Click and Script)** protocol type from the **ERP** category. To begin recording, click the **Record** button and perform typical actions in your SAP Web application. You should record your sign-in information in the **vuser_init** section, and the sign off process in the **vuser_end** section. For further information about creating and recording a script, see "Recording with VuGen" in *Volume I-Using VuGen*.

You can set event related recording options. See Chapter 17, "Click and Script Recording" in *Volume I-Using VuGen* for more information.

If you require a lower level script, or if you need to record on unsupported SAP control, use the **SAP-Web** protocol in the ERP category.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding SAP (Click and Script) Scripts

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported SAP objects, VuGen generates a function with an **sap_xxx** prefix.

In the following example, a user selected the **User Profile** tab. VuGen generated a **sap_portal** function.

```
web_browser("Close_2",
  "Snapshot=t7.inf",
  DESCRIPTION,
  "Ordinal=2",
  ACTION,
  "UserAction=Close",
  LAST);

lr_think_time(7);

web_text_link("Personalize",
  "Snapshot=t8.inf",
  DESCRIPTION,
  "Text=Personalize",
  ACTION,
  "UserAction=Click",
  LAST);

lr_think_time(6);

sap_portal("Sap Portal_2",
  "Snapshot=t9.inf",
  DESCRIPTION,
  "BrowserOrdinal=2",
  ACTION,
  "DetailedNavigation=User Profile",
  LAST);
```

Note: When you record a SAP (Click and Script) session, VuGen generates standard Web (Click and Script) functions for objects that are not SAP-specific. You do not need to explicitly specify the Web protocol. In the example above, VuGen generated a **web_text_link** function when the user clicked the **Personalize** button.

49

SAP-Web Protocol

You use VuGen's SAP-Web Vuser type, to record the activity in SAP Workplace or SAP Portal clients.

This chapter includes:

- ▶ About Developing SAP-Web Vuser Scripts on page 736
- ▶ Creating a SAP-Web Vuser Script on page 736
- ▶ Understanding a SAP-Web Vuser Script on page 738
- ▶ Replaying a SAP-Web Vuser Script on page 740

About Developing SAP-Web Vuser Scripts

You use VuGen to record typical SAP business processes. VuGen records SAP Workplace or Portal activity during the business processes, and generates a Vuser script. When you perform actions within your browser, VuGen generates functions that describe this activity. Each function begins with a **web** prefix.

During replay, these functions emulate user activity on the SAP Workplace or Portal clients. For example, `web_url` navigates to the PageBuilder.

```
web_url("PageBuilder[myPage]",
"URL=http://sonata.hplab.com/hrnp$30001/sonata.hplab.co.il:80/Action/PageBuilder[m
yPage]?pageName=com.sapportals.pct.home.mynews",
"Resource=0",
"RecContentType=text/html",
"Referer=http://sonata.hplab.co.il/sapportal",
"Snapshot=t2.inf",
"Mode=HTML",
EXTRARES,
"Url=/irj/services/laf/themes/portal/sap_mango_polarwind/..., ENDITEM,
LAST);
```

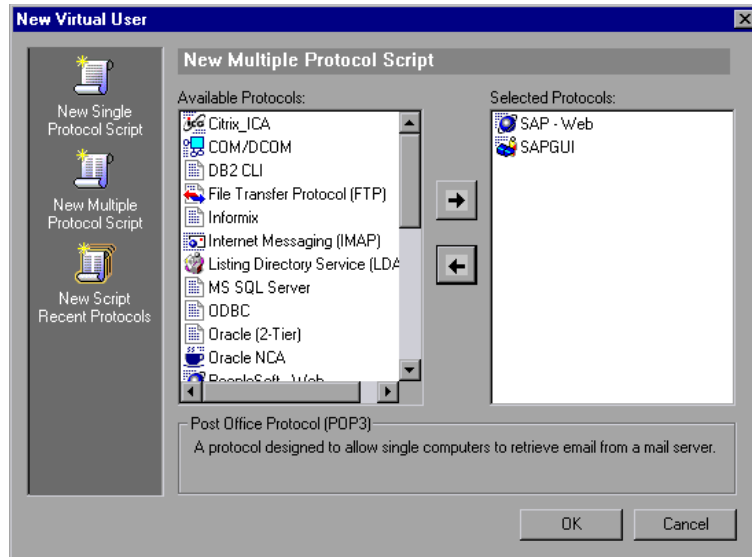
Creating a SAP-Web Vuser Script

The first step in creating a SAP-Web Vuser script, is choosing the Vuser and script type. The **SAP-Web** Vuser is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script. In addition, you can use the single-protocol SAP (Click and Script) Vuser type. The SAP (Click and Script) Vuser generates a higher level, more intuitive script.

To create a SAP-Web Vuser:

- 1** Invoke VuGen and select **File > New**.
- 2** To record a session that does not incorporate any SAPGUI controls within the Workplace or Portal clients, create a single-protocol Vuser script using the **SAP-Web** Vuser type.

- 3 To record a session that uses SAPGUI controls, create either:
- ▶ a single-protocol Vuser script, specifying the **SAP (Click and Script)** protocol.
 - ▶ a multi-protocol Vuser script, specifying both the **SAP-Web** and **SAPGUI** Vuser types.



Understanding a SAP-Web Vuser Script

The SAP-Web Vuser script typically contains several SAP transactions which make up a business process. The business process consists of functions that emulate user actions. For information about these functions, see the Web functions in the *Online Function Reference* (**Help > Function Reference**).

The following example shows a typical recording for a SAP Portal client:

```
vuser_init()
{
    web_reg_find("Text=SAP Portals Enterprise Portal 5.0",
                LAST);

    web_set_user("junior{UserNumber}",
                lr_decrypt("3ed4cfe457afe04e"),
                "sonata.hplab.com:80");

    web_url("sapportal",
            "URL=http://sonata.hplab.com/sapportal",
            "Resource=0",
            "RecContentType=text/html",
            "Snapshot=t1.inf",
            "Mode=HTML",
            EXTRARES,

            "Uri=/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,
            "Uri=/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 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/hrmp$30000/sonata.hplab.com:1000/Action/dummy?PASS_PARAMS=YES&dummyComp=dummy&Tcode=VA01&draggable=0&CompFName=VA01&Style=sap_mango_polarwind",
  "Resource=0",
  "RecContentType=text/html",
  "Referer=http://sonata.hplab.com/sapportal",
  "Snapshot=t9.inf",
  "Mode=HTML",
  LAST);

sapgui_open_connection_ex("/H/Protector/S/3200 /WP",
  "",
  "con[0]");

sapgui_select_active_connection("con[0]");

sapgui_select_active_session("ses[0]");

/*Before running script, enter password in place of asterisks in logon function*/

sapgui_logon("JUNIOR{UserNumber}",
  "ides",
  "800",
  "EN",
  BEGIN_OPTIONAL,
  "AdditionalInfo=sapgui102",
  END_OPTIONAL);

```

Replaying a SAP-Web Vuser Script

After creating and enhancing your SAP-Web Vuser script, you configure its run-time settings and run it from VuGen to check its functionality.

Run-Time settings let you control the Vuser behavior during replay. You configure these settings before running the Vuser script. You can set both General and Web related run-time settings.

The General settings include the run logic, pacing, logging, think time, and performance preferences. For information about the General run-time settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*. For SAP-Web specific settings, see Chapter 23, "Configuring Network Run-Time Settings" in *Volume I-Using VuGen*.

Once you configure the Run-Time settings, you save the Vuser script and run it from VuGen as a standalone test, to verify that it runs correctly. For further information, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After verifying that your Vuser script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller, Performance Center, or HP Business Availability Center* documentation.

50

Siebel-Web Protocol

You use VuGen to record the activity in a Siebel Web environment and generate a Vuser script. When you run the script, Vusers emulate the actions within your Siebel environment.

This chapter includes:

- ▶ About Developing Siebel-Web Vuser Scripts on page 741
- ▶ Recording a Siebel-Web Session on page 742
- ▶ Correlating Siebel-Web Scripts on page 743
- ▶ Correlating SWECOUNT, ROWID, and SWET Parameters on page 750
- ▶ Troubleshooting Siebel-Web Vuser Scripts on page 752

About Developing Siebel-Web Vuser Scripts

The Siebel-Web protocol is similar to the standard Web Vuser, with several changes in the default settings to allow it to work with the Siebel Customer Relationship Management (CRM) application.

You record typical activities in your Siebel session. VuGen records the actions and generates functions with a `web_` prefix, that emulate the actions.

The sections below provide tips for working with Siebel-Web recorded Vuser Scripts and provide samples of the parameters that need to be correlated.

Recording a Siebel-Web Session

When recording a Siebel-Web session, use the following guidelines:

To record a Siebel-Web Vuser script:

- 1** Create a Siebel-Web type Vuser script from the ERP category.
- 2** Set the following Recording Options:
 - ▶ Record node: **HTML based script**
 - Advanced HTML - Script options: **a script containing explicit URLs only**
 - Advanced HTML - Non HTML-generated elements: **Do not record**
 - ▶ Advanced node: Clear the **Reset context for each action** option.
- 3** Record the login in the **vuser_init** section.
- 4** Record the Business Process in **Action1**.
- 5** Record the logout in the **vuser_end** section.
- 6** In the Run-Time settings, clear the **Simulate a new user on each iteration** option in the Browser Emulation node.

For more information on how to configure the Recording Options and Web related Run-Time settings, see "Web, Wireless, and Oracle NCA Recording Options" on page 332 in *Volume I-Using VuGen*, and Chapter 23, "Configuring Network Run-Time Settings" in *Volume I-Using VuGen*.

Correlating Siebel-Web Scripts

When creating a test script for a Siebel session, you will most probably need to use correlation in your script. Correlation is the mechanism by which VuGen saves dynamic values to parameters during record and replay, for use at a later point in the script. If you replayed the recorded script as is, without correlation, it would fail, since the values of the arguments differ each time the script runs. An example of such variables are **SWECOUNT** and **SWEBMC**.

When you use correlation, VuGen saves the dynamic variables during both record and replay, and uses them at the appropriate points within the script. You can instruct VuGen to apply correlation during recording using one of the following methods:

► Siebel Correlation Library

The Siebel correlation library automatically correlates most of the dynamic values, creating a concise script that you can replay without major modifications. This is the recommended method for correlation.

► VuGen Native Siebel Correlation

The native, built-in rules, work on a low level, allowing you to debug your script and understand the correlations in depth.

Siebel Correlation Library

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

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

To enable correlation with this library:

- 1 Copy the DLL file into the bin directory of the product installation.
- 2 Open a multi-protocol script using the **Siebel-Web** Vuser type.

- 3 Enable UTF-8 support in the recording options. For more information, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.
- 4 Open the recording option's Correlation node and click **Import**. Import the rules file, **WebSiebel77Correlation.cor**, from the `\dat\webrulesdefaultsetting` directory. If you are prompted with warnings, click **Override**. For more information, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

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

When using the Siebel correlation library, verify that the SWE count rules (where the left boundary contains the **SWEC** string) are not disabled. For more information, see "Disabling and Enabling Rules" on page 748.

VuGen Native Siebel Correlation

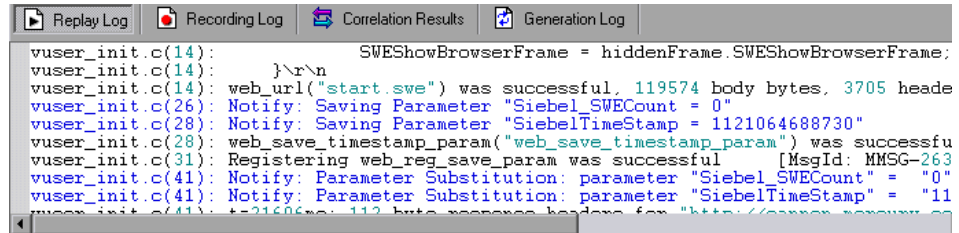
VuGen's native built-in rules for the Siebel server detect the Siebel server variables and strings, automatically saving them for use at a later point within the script.

You can view these rules in the list of correlation rules (see "Using VuGen's Correlation Rules" on page 634). The rules list the boundary criteria that are unique for Siebel server strings.

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

You can also create your own rules by entering unique boundary criteria in the Correlation Recording Options (see Chapter 42, "Web (HTTP/HTML) Correlation Rules") or after the recording from the Correlation Results tab (see "Performing a Scan for Correlations" on page 651).

In the Replay Log tab, VuGen indicates when it registers, saves, or uses the parameters. Note that to display this information, you need to enable Extended logging. For more information, see Chapter 22, "Configuring the Log Run-Time Settings" in *Volume I-Using VuGen*.



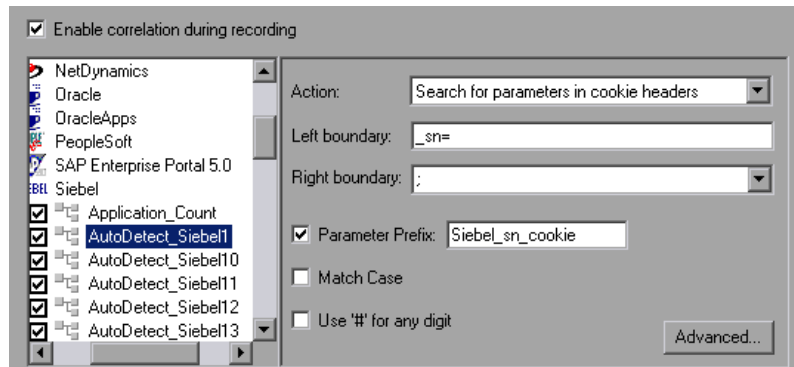
```

vuser_init.c(14): SWEShowBrowserFrame = hiddenFrame.SWEShowBrowserFrame;
vuser_init.c(14): }r\n
vuser_init.c(14): web_url("start.swe") was successful, 119574 body bytes, 3705 heade
vuser_init.c(26): Notify: Saving Parameter "Siebel_SWESCount = 0"
vuser_init.c(28): Notify: Saving Parameter "SiebelTimeStamp = 1121064688730"
vuser_init.c(28): web_save_timestamp_param("web_save_timestamp_param") was successfu
vuser_init.c(31): Registering web_reg_save_param was successful [MsgId: MMSG-263
vuser_init.c(41): Notify: Parameter Substitution: parameter "Siebel_SWESCount" = "0"
vuser_init.c(41): Notify: Parameter Substitution: parameter "SiebelTimeStamp" = "11
vuser_init.c(41): t=216866: 112 bytes response headers for "http://www.siebel.com"

```

Simple Variable Correlation

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



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

```

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

...

web_url("start.swe_3",
"URL=http://cannon.hplab.com/callcenter_enu/start.swe?SWECmd=GotoPostedAction
&SWEDIC=true&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECCount}&SWEFrame
=top._sweclient&SWECS=true",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://cannon.hplab.com/callcenter_enu/start.swe?SWECmd=GetCachedFra
me&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECCount}&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=SWECKL", "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 749.

Disabling and Enabling Rules

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

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

To disable rules:

- 1 Open the Correlation recording options. Select **Tools > Recording Options** and click the Correlation node.
- 2 Select the **Enable correlation during recording** option. The dialog box displays the supported servers.
- 3 Expand the rules under Siebel and view their properties.
- 4 Clear the check box adjacent to the rule for each rule you want to disable.

SWEC Correlation

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

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

or

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

VuGen handles the changes of the SWEC by incrementing a counter before each relevant step. VuGen stores the current value of the SWEC in a separate variable (`Siebel_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.

Correlating SWECCount, ROWID, and SWET Parameters

This section provides tips for correlating several special parameters:

- ▶ SWECCount
- ▶ Row ID Length
- ▶ SWETS (Timestamps)

SWECCount

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.

Row ID Length

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

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

If you define parameters to correlate the string as

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

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

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

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

SWETS (Timestamps)

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

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

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

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

Troubleshooting Siebel-Web Vuser Scripts

This section provides information about errors you might encounter when creating a script, and the breakdown diagnostic tool.

- ▶ Typical Errors
- ▶ Recording Breakdown Information

Typical Errors

You may encounter one or more of the following errors while creating a Siebel-Web Vuser script:

- ▶ Back or Refresh Error
- ▶ Same Values
- ▶ No Content HTTP Response
- ▶ Restoring the Context
- ▶ Cannot Locate Record
- ▶ End of File
- ▶ Unable to Retrieve Search Categories

Back or Refresh Error

An error message relating to **Back or Refresh** typically has the following text:

We are unable to process your request. This is most likely because you used the browser BACK or REFRESH button to get to this point.

Cause: The possible causes of this problem may be:

- ▶ The SWEC was not correlated correctly for the current request.
- ▶ The SWETS was not correlated correctly for the current request.
- ▶ The request was submitted twice to the Siebel server without the SWEC being updated.

- A previous request should have opened a frame for the browser to download. This frame was not created on the server probably because the SWEMethod has changed since the recording.

Same Values

A typical Web page response to the **Same Values** error is:

```
@0'0'3'3''0'UC'1`Status`Error`SWEC`10'0'1`Errors`0'2'0`Level0'0`ErrMsg`Th
e same values for 'Name' already exist. If you would like to enter a new record,
please make sure that the field values are unique.`ErrCode`28591`
```

Cause: The possible cause of this problem may be that one of the values in the request (in the above example, the value of the Name field) duplicates a value in another row of the database table. This value needs to be replaced with a unique value to be used for each iteration per user. The recommended solution is to replace the row ID with its parameter instead insuring that it will be unique.

No Content HTTP Response

A typical HTTP response for a **No Content HTTP Response** type error is:

```
HTTP/1.1 204 No Content
Server: Microsoft-IIS/5.0
Date: Fri, 31 Jan 2003 21:52:30 GMT
Content-Language: en
Cache-Control: no-cache
```

Cause: The possible causes of this problem may be that the row ID is not correlated at all or that it is correlated incorrectly.

Restoring the Context

The typical Web page response to the **Restoring the Context** type error is:

```
@0'0'3'3''0'UC'1`Status`Error`SWEC`9'0'1`Errors`0'2'0`Level0'0`ErrMsg`An
error happened during restoring the context for requested
location`ErrCode`27631`
```

Cause: The possible causes of this problem may be that the rowid is not correlated or that it is correlated incorrectly.

Cannot Locate Record

The typical Web page response to the **Cannot locate record** type error is:

```
@0'0'3'3''0'UC`1`Status`Error`SWEC`23`0`2`Errors`0`2`0`Level0`0`ErrMsg`Ca  
nnot locate record within view: Contact Detail - Opportunities View applet:  
Opportunity List Applet.`ErrCode`27573`
```

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

End of File

The typical Web page response to the **End of File** type error is:

```
@0'0'3'3''0'UC`1`Status`Error`SWEC`28`0`1`Errors`0`2`0`Level0`0`ErrMsg`An  
end of file error has occurred. Please continue or ask your systems administrator  
to check your application configuration if the problem persists.`ErrCode`28601`
```

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

Unable to Retrieve Search Categories

The typical Web page response to the **Unable to Retrieve Search Categories** type error is:

Cause: A possible cause of this problem may be that the search frame was not downloaded from the server. This occurs when the previous request did not ask the server to create the search frame correctly.

Recording Breakdown Information

VuGen provides a diagnostic tool for understanding the transaction components in your test—**transaction breakdown**. Using transaction breakdown, you can determine where the bottlenecks are and the issues that need to be resolved.

When preparing your script for transaction breakdown, we recommend that you add think time at the end of each transaction using the ratio of one second per hour of testing. For more information on entering think time, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

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.

51

RTE Protocol

RTE Vusers operate terminal emulators in Windows environments. This chapter describes how to develop Windows-based RTE Vuser scripts.

This chapter includes:

- ▶ About Developing RTE Vuser Scripts on page 757
- ▶ Introducing RTE Vusers on page 758
- ▶ Understanding RTE Vuser Technology on page 759
- ▶ Getting Started with RTE Vuser Scripts on page 759
- ▶ Using TE Functions on page 761
- ▶ Working with Ericom Terminal Emulation on page 761
- ▶ Mapping Terminal Keys to PC Keyboard Keys on page 763

About Developing RTE Vuser Scripts

RTE Vusers operate terminal emulators in order to load test client/server systems.

You record a terminal emulator session with VuGen to represent a true user's actions. You can then enhance your recorded script with transaction and synchronization functions.

This chapter describes how to develop Windows-based RTE Vuser scripts.

Introducing RTE Vusers

An RTE Vuser types character input into a terminal emulator, submits the data to a server, and then waits for the server to respond. For instance, suppose that you have a server that maintains customer information for a maintenance company. Every time a field service representative makes a repair, he accesses the server database by modem using a terminal emulator. The service representative accesses information about the customer and then records the details of the repair that he performs.

You could use RTE Vusers to emulate this case. An RTE Vuser would:

- 1** Type **60** at the command line to open an application program.
- 2** Type **F296**, the field service representative's number.
- 3** Type **NY270**, the customer number.
- 4** Wait for the word "Details" to appear on the screen. The appearance of "Details" indicates that all the customer details are displayed on the screen.
- 5** Type **Changed gasket P249, and performed Major Service** the details of the current repair.
- 6** Type **Q** to close the application program.

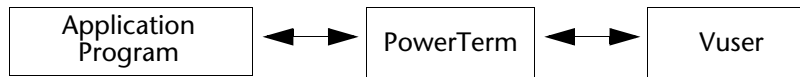
You use VuGen to create RTE Vuser scripts. The script generator records the actions of a human user in a terminal emulator. It records the keyboard input from the terminal window, generates the appropriate statements, and inserts them into the Vuser script. While you record, the script generator automatically inserts synchronization functions into the script. For details, see Chapter 53, "RTE - Synchronization."

Understanding RTE Vuser Technology

An RTE Vuser emulates the actions of a real user. Human users use terminals or terminal emulators to operate application programs.



In the RTE Vuser environment, a Vuser replaces the human. The Vuser operates PowerTerm, a terminal emulator.



PowerTerm works like a standard terminal emulator, supporting common protocols such as IBM 3270 & 5250, VT100, VT220, and VT420-7.

Getting Started with RTE Vuser Scripts

This section provides an overview of the process of developing RTE Vuser scripts using VuGen.

To develop an RTE Vuser script:

1 Record the basic script using VuGen.

Use the Virtual User Generator (VuGen) to record the operations that you perform in a terminal emulator. VuGen records the keyboard input from the terminal window, generates the appropriate statements, and then inserts these statements into the Vuser script.

For details, see Chapter 52, "RTE - Recording."

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, synchronization functions, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

4 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

5 Run the script from VuGen.

Run the script from VuGen to verify that it runs correctly. View the standard output to verify that the program is communicating properly with the server.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using TE Functions

The functions developed to emulate a terminal communicating with a server are called TE Vuser functions. Each TE Vuser function has a **TE** prefix. VuGen automatically records most of the TE functions listed in this section during an RTE session. You can also manually program any of the functions into your script.

The TE functions are divided into the following categories: Terminal Emulator Connection, Text Retrieval, Cursor, System Variable, Error Code, Typing, and Synchronization Functions.

You can also manually program any of the functions into your Vuser script. In text view, you can manually add new functions utilizing the Intellisense and Complete Function features. In Tree view, select **Insert > New Step** and select the desired step.

For syntax and examples of the TE functions, see the *Online Function Reference* (**Help > Function Reference**).

Working with Ericom Terminal Emulation

VuGen supports record and replay with Ericom Terminal Emulators.

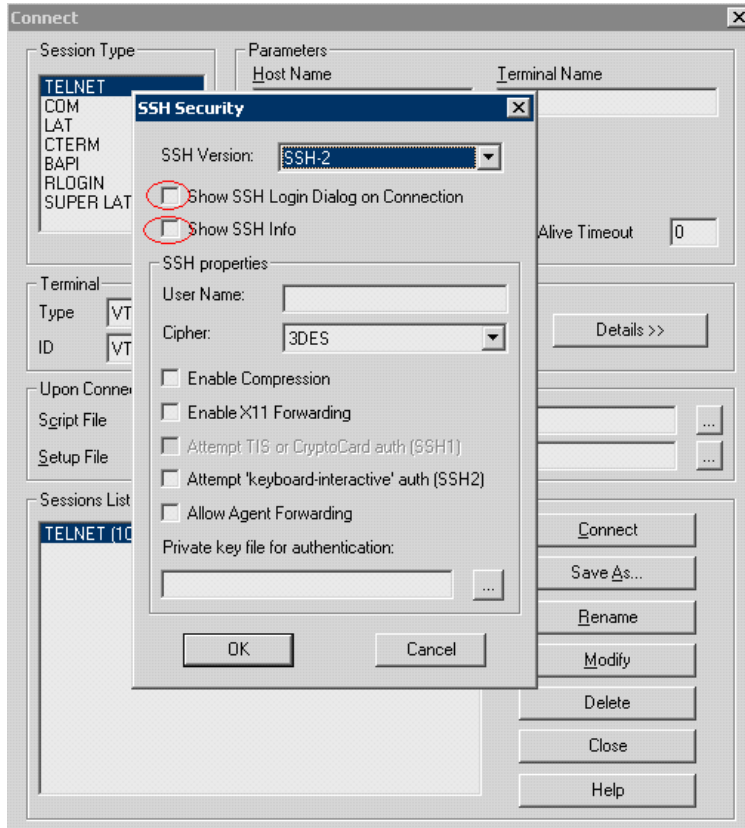
The Ericom support handles escape sequences during record and replay. Ericom's PowerTerm lets you map PC keys to custom escape sequences. For information about mapping, see the PowerTerm help.

When a user presses mapped keys while recording an Ericom VT session, VuGen generates **TE_send_text** functions instead of the standard **TE_type**. This allows the script to handle custom escape sequences in a single step. For more information, see the *Online Function Reference* (**Help > Function Reference**) for the **TE_send_text** function.

SSL and SSH Support for Ericom

VuGen also supports SSL/SSH record and replay for the RTE Ericom library. To work with SSL or SSH, you select the type in the **Security** section of the Connect dialog box.

When working with SSH Security, by default VuGen opens a popup dialog box prompting you for more information. We recommend that you disable the **Show** options to prevent the popups from being issued. If you enable these popups, it may affect the replay. You can access the advanced security options by clicking the **Details** button.



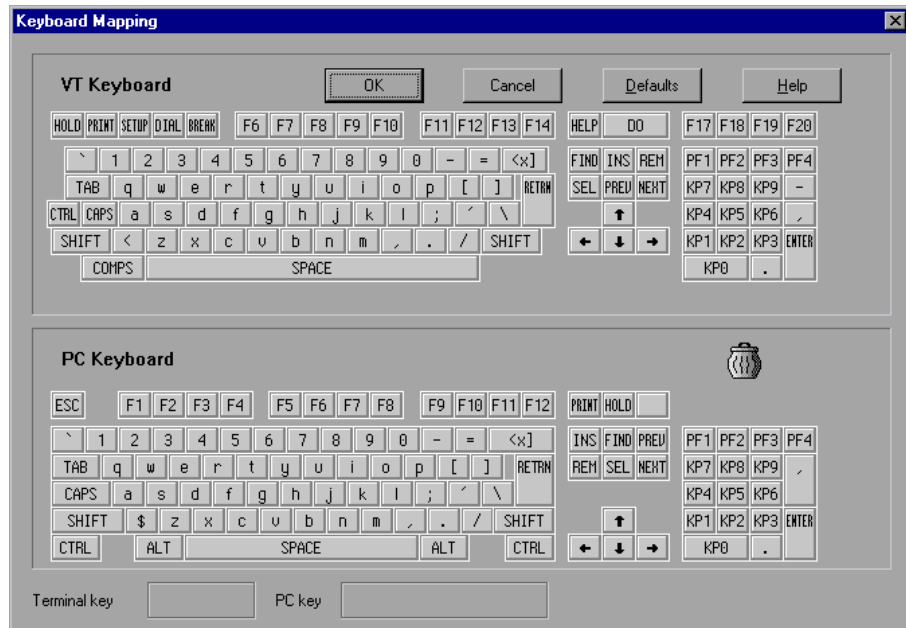
Mapping Terminal Keys to PC Keyboard Keys

Because you are using a terminal emulator, you will be using a PC keyboard in place of a terminal keyboard. Many keys that are found on the terminal keyboard are not available on a PC keyboard. Examples of such keys are HELP, AUTOR, and PUSH, which are found on the IBM 5250 keyboard. To successfully operate the terminal emulator and any associated application programs, you may have to map certain terminal keys to keys on the PC keyboard.

To map a terminal key to a key on the PC keyboard:



- 1 In the terminal emulator, select **Options > Keyboard Map**, or click the **Keyboard Mapping** button. The Keyboard Mapping dialog box opens.



- 2 Click the Keyboard **Mapping** button on the toolbar. To map a terminal key to a PC key, drag a key from the upper terminal keyboard to a PC key on the lower keyboard.

You can click the Shift and/or Control keys on the upper keyboard to display additional key functions that can be viewed only by first selecting either of these keys. You can then drag the required key from the upper terminal keyboard to a key on the lower PC keyboard.

To cancel a definition, drag the PC key definition to the wastebasket. This restores the default function of the PC key.

To restore the default mappings, click **Defaults**.

52

RTE - Recording

You use VuGen to record Windows-based Remote Terminal Emulation (RTE) Vuser scripts.

This chapter includes:

- ▶ About Recording RTE Vuser Scripts on page 766
- ▶ Creating a New RTE Vuser Script on page 766
- ▶ Recording the Terminal Setup and Connection Procedure on page 767
- ▶ Recording Typical User Actions on page 771
- ▶ Recording the Log Off Procedure on page 772
- ▶ Typing Input into a Terminal Emulator on page 772
- ▶ Generating Unique Device Names on page 775
- ▶ Setting the Field Demarcation Characters on page 776

About Recording RTE Vuser Scripts

You use VuGen to record Windows-based RTE Vuser scripts. VuGen uses the PowerTerm terminal emulator to emulate a wide variety of terminal types. You use PowerTerm to perform a typical terminal connection, followed by typical business processes. Thereafter, you perform the log off procedure. While you perform typical user actions in the terminal emulator, VuGen generates the appropriate statements, and inserts them into a Vuser script. You can view and edit the script while recording.

Before recording an RTE Vuser script, make sure that the recording options are set correctly. The recording options allow you to control how VuGen generates certain functions while you record a Vuser script. VuGen applies the recording options during all subsequent recording sessions.

Creating a New RTE Vuser Script

Before recording a user's actions into a Vuser script, you must open one. You can open an existing script, or create a new one. You use VuGen to create a new Vuser script.

To create a new RTE Vuser script:

1 Select **Virtual User Generator** from your product's start menu. The VuGen window opens.



2 Click the **New** button. The **New Virtual User** dialog box opens:

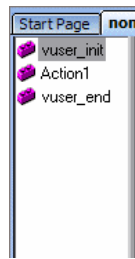
3 Select the **Legacy** protocol type, and select **Terminal Emulator (RTE)**. Click **OK**. VuGen generates and displays an empty RTE script, with the cursor positioned to begin recording in the **vuser_init** section.

Recording the Terminal Setup and Connection Procedure

After you create a skeleton Vuser script, you record the terminal setup and connection procedure into the script. VuGen uses the PowerTerm terminal emulator when you record an RTE Vuser script.

To record the terminal setup and connection procedure:

- 1 Open an existing RTE Vuser script, or create a new one.
- 2 In the **Sections** box, select the section into which you want VuGen to insert the recorded statements. The available sections are **vuser_init**, **Actions**, and **vuser_end**.



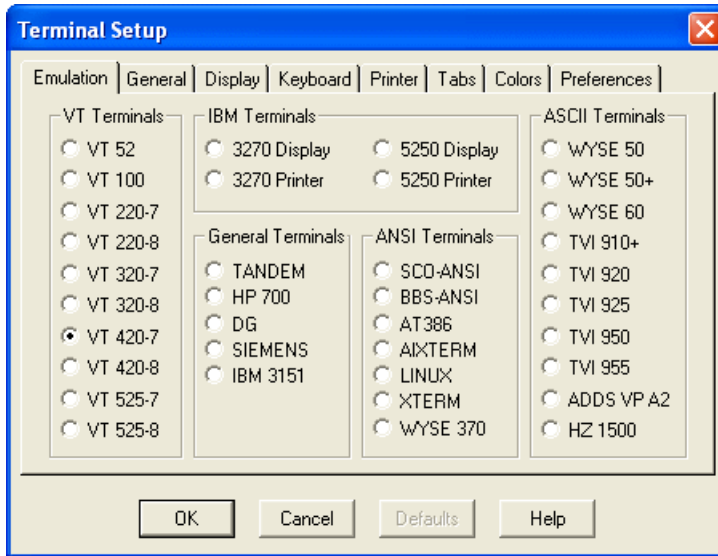
Note: Always record the terminal setup and connection procedure into the **vuser_init** section. The **vuser_init** section is not repeated when you run multiple iterations of a Vuser script—only the **Actions** section is repeated. For more information on the iteration settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

- 3 In the Vuser script, place the cursor at the location where you want to begin recording.



- 4 Click the **Record** button. The PowerTerm main window opens.

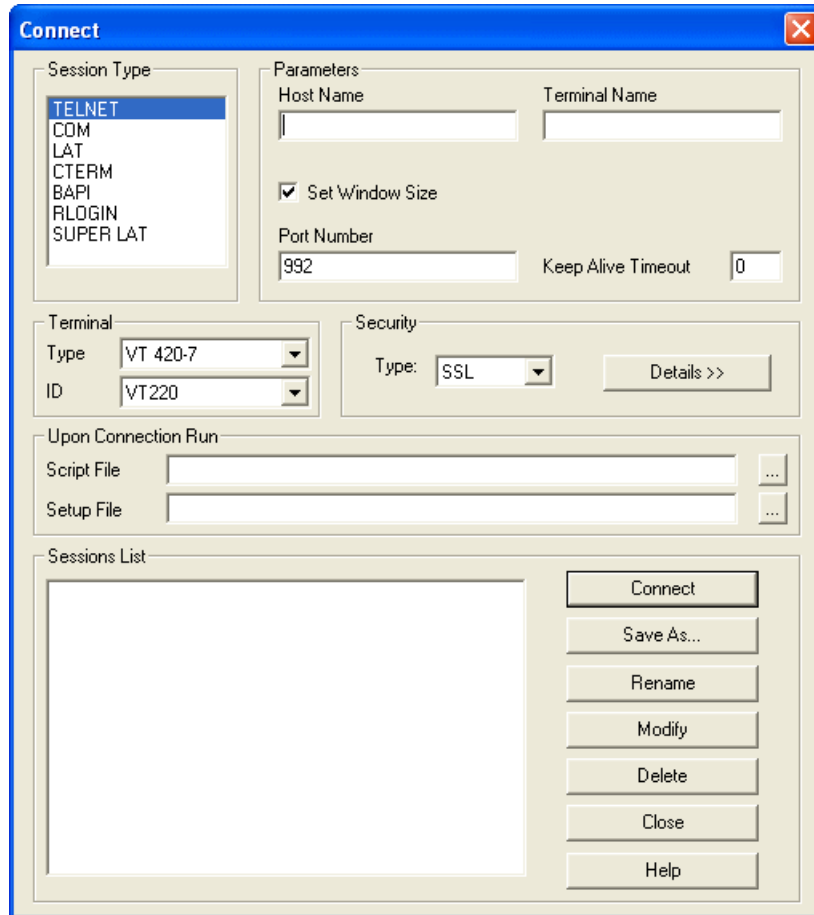
- 5 From the PowerTerm menu bar, select **Terminal > Setup** to display the Terminal Setup dialog box.



- 6 Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.

Note: Select an IBM terminal type to connect to an AS/400 machine or an IBM mainframe; select a VT terminal type to connect to a UNIX workstation.

7 Select **Communication > Connect** to display the Connect dialog box.



- 8 Under **Session Type**, select the type of communication to use.
- 9 Under **Parameters**, specify the required options. The available parameters vary depending on the type of session that you select. For details on the parameters, click **Help**.

Tip: Click **Save As** to save the parameter-sets for re-use in the future. The parameter-sets that you save are displayed in the Sessions List box.

- 10** Click **Connect**. PowerTerm connects to the specified system, and VuGen inserts a **TE_connect** function into the script, at the insertion point. The **TE_connect** statement has the following form:

```
/* *** The terminal type is VT 100. */  
TE_connect(  
    "comm-type = telnet;"  
    "host-name = alfa;"  
    "telnet-port = 992;"  
    "terminal-id = ;"  
    "set-window-size = true;"  
    "security-type = ssl;"  
    "ssl-type = tls1;"  
    "terminal-type = vt100;"  
    "terminal-model = vt100;"  
    "login-command-file = ;"  
    "terminal-setup-file = ;"  
    , 60000);  
if (TE_errno != TE_SUCCESS)  
    return -1;
```

The inserted **TE_connect** statement is followed by an if statement that checks whether or not the **TE_connect** function succeeds during replay.

Note: Do not record more than one connection to a server (**TE_connect**) in a Vuser script.

The terminal setup and connection procedure is complete. You are now ready to begin recording typical user actions into the Vuser script, as described below.

Recording Typical User Actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the **Actions** section of the Vuser script. Only the **Actions** section of a Vuser script is repeated when you run multiple iterations of the script. For details on setting iterations, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

When recording a session, VuGen records the text strokes and not the text. Therefore, it is not recommended that you copy and paste commands into the PowerTerm window—instead, type them in directly.

To record user actions:

- 1** Open an existing RTE Vuser script, and then click **Actions** in the **Section** box.
- 2** Proceed to perform typical user actions in the terminal emulator. VuGen generates the appropriate statements, and inserts them into the Vuser script while you type. If necessary, you can edit the recorded statements while you record the script.



Note: By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function. To change the waiting time, see Chapter 16, "Recording Options for Selected Protocols" in *Volume I-Using VuGen*.

When you finish recording the typical user actions, proceed to record the log off procedure, as described in the next section.

Recording the Log Off Procedure

You record the Vuser log off process into the **vuser_end** section of the Vuser script. The **vuser_end** section is not repeated when you run many iterations of the script. For details on setting iterations, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

To record the log off procedure:

- 1** Make sure that you have performed and recorded the typical user actions as described in the previous section.
- 2** In the VuGen main window, click **vuser_end** in the **Section** box.
- 3** Perform the log off procedure. VuGen records the procedure into the **vuser_end** section of the script.
-  **4** Click **Stop Recording on** the Recording toolbar. The main VuGen window displays all the recorded statements.
-  **5** Click **Save to** save the recorded session. The Save As dialog box opens (for new Vuser scripts only). Specify a script name. After recording a script, you can manually edit it in VuGen's main window.

Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to "type" character input into the PowerTerm terminal emulator:

- **TE_type** sends characters to the terminal emulator. When recording, the VuGen automatically generates **TE_type** functions for keyboard input to the terminal window. For details, see "Using the TE_type Function" on page 773.
- **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 774. Alternatively, you can set the typing style by using the run-time settings. See Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen* for details.

Note: While recording an RTE Vuser script, do not use the mouse to relocate the cursor within the terminal emulator window. VuGen does not record these cursor movements.

Using the TE_type Function

When you record a script, the VuGen records all keyboard input and generates appropriate **TE_type** functions. During execution, **TE_type** functions send formatted strings to the terminal emulator.

Keyboard input is defined as a regular text string (including blank spaces). For example:

```
TE_type("hello, world");
```

Input key names longer than one character are represented by identifiers beginning with the letter *k*, and are bracketed within greater-than/less-than signs (< >).

For example, the following function depicts the input of the Return key followed by the Control and y keys:

```
TE_type("<kReturn><kControl-y>");
```

Some other examples include: <kF1>, <kUp>, <kF10>, <kHelp>, <kTab>.

To determine a key name, record an operation on the key, and then check the recorded statement for its name.

Note: When you program a **TE_type** statement (rather than recording it), use the key definitions provided in the *Online Function Reference* (**Help > Function Reference**).

Setting the Timeout Value for TE_type

If a Vuser attempts to submit a **TE_type** statement while the system is in X SYSTEM (or input inhibited) mode, the Vuser will wait until the X SYSTEM mode ends before typing. If the system stays in X SYSTEM mode for more than **TE_XSYSTEM_TIMEOUT** milliseconds, then the **TE_type** function returns a **TE_TIMEOUT** error.

You can set the value of **TE_XSYSTEM_TIMEOUT** by using **TE_setvar**. The default value for **TE_XSYSTEM_TIMEOUT** is 30 seconds.

Allowing a Vuser to Type Ahead

Under certain circumstances you may want a Vuser to submit a keystroke even though the system is in X SYSTEM (or input inhibited) mode. For example, you may want the Vuser to press the **Break** key. You use the **TE_ALLOW_TYPEAHEAD** variable to enable the Vuser to submit a keystroke even though the system is in X SYSTEM mode.

Set **TE_ALLOW_TYPEAHEAD** to zero to disable typing ahead, and to any non-zero number to permit typing ahead. You use **TE_setvar** to set the value of **TE_ALLOW_TYPEAHEAD**. By default, **TE_ALLOW_TYPEAHEAD** is set to zero, preventing keystrokes from being sent during X SYSTEM mode.

For more information about the **TE_type** function and its conventions, see the *Online Function Reference (Help > Function Reference)*.

Setting the Typing Style

You can set two typing styles for RTE Vuser: **FAST** and **HUMAN**. In the **FAST** style, the Vuser types input into the terminal emulator as quickly as possible. In the **HUMAN** style, the Vuser pauses after typing each character. In this way, the Vuser more closely emulates a human user typing at the keyboard.

You set the typing style using the **TE_typing_style** function. The syntax of the **TE_typing_style** function is:

```
int TE_typing_style (char *style);
```

where *style* can be **FAST** or **HUMAN**. The default typing style is **HUMAN**. If you select the **HUMAN** typing style, the format is:

HUMAN, *delay* [,*first_delay*]

The delay indicates the interval (in milliseconds) between keystrokes. The optional parameter *first_delay* indicates the wait (in milliseconds) before typing the first character in the string. For example,

```
TE_typing_style ("HUMAN, 100, 500");
TE_type ("ABC");
```

means that the Vuser will wait 0.5 seconds before typing the letter A; it will then wait 0.1 seconds before typing "B" and then a further 0.1 seconds before typing "C".

For more information about the **TE_typing_style** function and its conventions, see the *Online Function Reference* (**Help > Function Reference**).

In addition to setting the typing style by using the **TE_typing_style** function, you can also use the run-time settings. For details, see Chapter 24, "RTE Run-Time Settings."

Generating Unique Device Names

Some protocols, such as APPC, require a unique device name for each terminal that logs on to the system. Using the run-time settings, you can specify that the **TE_connect** function generate a unique 8-character device name for each Vuser, and connect using this name. Although this solves the requirement for uniqueness, some systems have an additional requirement: The device names must conform to a specific format. See Chapter 22, "Configuring Run-Time Settings" in *Volume I-Using VuGen* 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**.

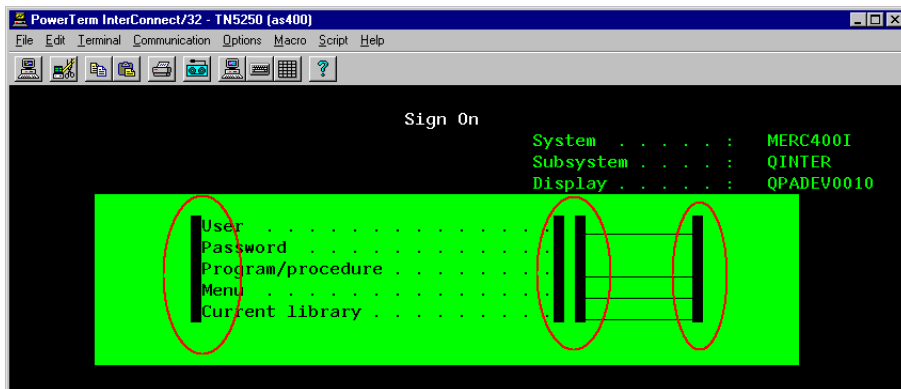
If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

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.

53

RTE - Synchronization

Synchronization functions in an RTE Vuser script help you synchronize the input that a Vuser submits to a terminal emulator with the responses from the server.

This chapter includes:

- About Synchronizing Vuser Scripts on page 779
- Synchronizing Block-Mode (IBM) Terminals on page 781
- Synchronizing Character-Mode (VT) Terminals on page 784

About Synchronizing Vuser Scripts

Depending on the system you are testing, you may need to synchronize the input that a Vuser sends to a terminal emulator with the subsequent responses from the server. When you synchronize input, you instruct the Vuser to suspend script execution and wait for a cue from the system, before the Vuser performs its next action. For instance, suppose that a human user wants to submit the following sequence of key strokes to a bank application:

- 1** Type 1 to select "Financial Information" from the menu of a bank application.
- 2** When the message "What information do you require?" appears, type 3 to select "Dow Jones Industrial Average" from the menu.
- 3** When the full report has been written to the screen, type 5 to exit the bank application.

In this example, the input to the bank application is synchronized because at each step, the human user waits for a visual cue before typing. This cue can be either the appearance of a particular message on the screen, or stability of all the information on the screen.

You can synchronize the input of a Vuser in the same way by using the TE-synchronization functions, `TE_wait_sync`, `TE_wait_text`, `TE_wait_silent`, and `TE_wait_cursor`. These functions effectively emulate a human user who types into a terminal window and then waits for the server to respond, before typing in the next command.

The `TE_wait_sync` function is used to synchronize block-mode (IBM) terminals only. The other TE-synchronization functions are used to synchronize character-mode (VT) terminals.

When you record an RTE Vuser script, VuGen can automatically generate and insert `TE_wait_sync`, `TE_wait_text`, and `TE_wait_cursor` statements into the script. You use VuGen's recording options to specify which synchronization functions VuGen should insert.

Note: Do not include any synchronization statements in the *Vuser_end* section of a Vuser script. Since a Vuser can be aborted at any time, you cannot predict when the *Vuser_end* section will be executed.

Synchronizing Block-Mode (IBM) Terminals

The `TE_wait_sync` function is used for synchronization RTE Vusers operating block-mode (IBM) terminals. Block-mode terminals display the "X SYSTEM" message to indicate that the system is in Input Inhibited mode. When a system is in the Input Inhibited mode no typing can take place because the terminal emulator is waiting for a transfer of data from the server.

When you record a script on a block-mode terminal, by default, VuGen generates and inserts a `TE_wait_sync` function into the script each time the "X SYSTEM" message appears. You use VuGen's recording options to specify whether or not VuGen should automatically insert `TE_wait_sync` functions.

When you run a Vuser script, the `TE_wait_sync` function checks if the system is in the X SYSTEM mode. If the system is in the X SYSTEM mode, the `TE_wait_sync` function suspends script execution. When the "X SYSTEM" message is removed from the screen, script execution continues.

Note: You can use the `TE_wait_sync` function only with IBM block-mode terminals emulators (5250 and 3270).

In general, the `TE_wait_sync` function provides adequate synchronization for all block-mode terminal emulators. However, if the `TE_wait_sync` function is ineffective in a particular situation, you can enhance the synchronization by including a `TE_wait_text` function. For more information on the `TE_wait_text` function, see "Waiting for Text to Appear on the Screen" on page 786, and the *Online Function Reference (Help > Function Reference)*.

The syntax of the `TE_wait_sync` function is:

```
TE_wait_sync ();
```

In the following script segment, the Vuser logs on with the user name "QUSER" and the password "HPLAB". The Vuser then presses **Enter** to submit the login details to the server. The terminal emulator displays the X SYSTEM message while the system waits for the server to respond.

The `TE_wait_sync` statement causes the Vuser to wait until the server has responded to the login request, that is, for the X SYSTEM message to be removed—before executing the next line of the script.

```
TE_type("QUSER");
lr_think_time(2);
TE_type("<kTab>HPLAB");
lr_think_time(3);
TE_type("<kEnter>");
TE_wait_sync();
....
```

When a `TE_wait_sync` function suspends the execution of a script while an X SYSTEM message is displayed, the Vuser continues to monitor the system—waiting for the X SYSTEM message to disappear. If the X SYSTEM message does not disappear before the synchronization timeout expires, the `TE_wait_sync` function returns an error code. The default timeout is 60 seconds.

To set the `TE_wait_sync` synchronization timeout:

- 1** Select **Vuser > Run-Time Settings**. The Run-Time Settings dialog box appears.
- 2** Select the **RTE:RTE** node in the Run-Time setting tree.
- 3** Under **X SYSTEM Synchronization**, enter a value (in seconds) in the **Timeout** box.
- 4** Click **OK** to close the Run-Time Settings dialog box.

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X SYSTEM mode. When the terminal returns from the X SYSTEM mode, the Vuser continues to monitor the system for a short period to verify that the terminal is fully stable, that is, that the system does not return to the X SYSTEM mode. Only then does the **TE_wait_sync** function terminate and allow the Vuser to continue executing its script. The period that the Vuser continues to monitor the system, after the system has returned from the X SYSTEM mode, is known as the stable time. The default stable time is 1000 milliseconds.

You may need to increase the stable time if your system exhibits the following behavior:

When a system returns from the X SYSTEM mode, some systems "flickers" to and from the X SYSTEM for a short period of time until the system stabilizes. If the system remains out of the X SYSTEM mode for more than one second, and then returns to the X SYSTEM mode, the **TE_wait_sync** function will assume that the system is stable. If a Vuser then tries to type information to the system, the system will shift into keyboard-locked mode.

Alternatively, if your system never flickers when it returns from the X SYSTEM mode, you can reduce the stable time to less than the default value of one second.

To change the stable time for TE_wait_sync functions:

- 1** Select **Vuser > Run-Time Settings**. The Run-Time Settings dialog box appears.
- 2** Select the **RTE:RTE** node.
- 3** Under **X SYSTEM Synchronization**, enter a value (in milliseconds) in the **Stable time** box.
- 4** Click **OK** to close the Run-Time Settings dialog box.

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

You can instruct VuGen to record the time that the system remains in the X SYSTEM mode each time that the X SYSTEM mode is entered. To do so, VuGen inserts a `TE_wait_sync_transaction` function after each `TE_wait_sync` function, as shown in the following script segment:

```
TE_wait_sync();  
TE_wait_sync_transaction("syncTrans1");
```

Each `TE_wait_sync_transaction` function creates a transaction with the name "default." This allows you to analyze how long the terminal emulator waits for responses from the server during a scenario run. You use the recording options to specify whether VuGen should generate and insert `TE_wait_sync_transaction` statements.

To instruct VuGen to insert `TE_wait_sync_transaction` statements:

- 1 Select **Vuser > Recording Options**. The Recording Settings dialog box appears.
- 2 Select the **Generate Automatic X SYSTEM transactions** option, and then click **OK**.

Synchronizing Character-Mode (VT) Terminals

There are three types of synchronization that you can use for character-mode (VT) terminals. The type of synchronization that you select depends on:

- the design of the application that is running in the terminal emulator
- the specific action to be synchronized

Waiting for the Cursor to Appear at a Specific Location

The preferred method of synchronization for VT type terminals is cursor synchronization. Cursor synchronization is particularly useful with full-screen or form-type applications, as opposed to scrolling or TTY-type applications.

Cursor synchronization uses the `TE_wait_cursor` function. When you run an RTE Vuser script, the `TE_wait_cursor` function instructs a Vuser to suspend script execution until the cursor appears at a specified location on the screen. The appearance of the cursor at the specified location means that the application is ready to accept the next input from the terminal emulator.

The syntax of the `TE_wait_cursor` function is:

```
int TE_wait_cursor (int col, int row, int stable, int timeout);
```

During script execution, the `TE_wait_cursor` function waits for the cursor to reach the location specified by *col*, *row*.

The **stable** parameter specifies the time (in milliseconds) that the cursor must remain at the specified location. If you record a script using VuGen, **stable** is set to 100 milliseconds by default. If the client application does not become stable in the time specified by the **timeout** parameter, the function returns TIMEOUT. If you record a script using VuGen, **timeout** is set by default to the value of TIMEOUT, which is 90 seconds. You can change the value of both the **stable** and **timeout** parameters by directly editing the recorded script.

The following statement waits for the cursor to remain stable for three seconds. If the cursor doesn't stabilize within 10 seconds, the function returns TIMEOUT.

```
TE_wait_cursor (10, 24, 3000, 10);
```

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

You can instruct VuGen to automatically generate `TE_wait_cursor` statements, and insert them into a script, while you record the script. The following is an example of a `TE_wait_cursor` statement that was automatically generated by VuGen:

```
TE_wait_cursor(7, 20, 100, 90);
```

To instruct VuGen to automatically generate `TE_wait_cursor` statements, and insert them into a script while recording:

- 1 Select **Vuser > Recording Options**. The Recording Settings dialog box appears.
- 2 Under **Generate Automatic Synchronization Commands** select the **Cursor** check box, and then click **OK**.

Waiting for Text to Appear on the Screen

You can use text synchronization to synchronize an RTE Vuser running on a VT terminal emulator. Text synchronization uses the `TE_wait_text` function. During script execution, the `TE_wait_text` function suspends script execution and waits for a specific string to appear in the terminal window before continuing with script execution. Text synchronization is useful with those applications in which the cursor does not consistently appear in a predefined area on the screen.

Note: Although text synchronization is designed to be used with character mode (VT) terminals, it can also be used with IBM block-mode terminals. Do not use automatic text synchronization with block-mode terminals.

The syntax of the `TE_wait_text` function is:

```
int TE_wait_text (char *pattern, int timeout, int col1, int row1, int col2, int row2,
                 int *retcol, int *retrow, char *match);
```

This function waits for text matching *pattern* to appear within the rectangle defined by col1, row1, col2, row2. Text matching the pattern is returned to **match**, and the actual row and column position is returned to **retcol** and **retrow**. If the **pattern** does not appear before the **timeout** expires, the function returns an error code. The **pattern** can include a regular expression. See the *Online Function Reference* for details on using regular expressions. Besides the **pattern** and **timeout** parameters, all the other parameters are optional.

If **pattern** is passed as an empty string, the function will wait for timeout if it finds any text at all within the rectangle. If there is no text, it returns immediately.

If the *pattern* does appear, then the function waits for the emulator to be stable (finish redrawing, and not display any new characters) for the interval defined by the `TE_SILENT_SEC` and `TE_SILENT_MILLI` system variables. This, in effect, allows the terminal to become stable and emulates a human user.

If the terminal does not become stable within the interval defined by `TE_SILENT_TIMEOUT`, script execution continues. The function returns 0 for success, but sets the `TE_errno` variable to indicate that the terminal was not silent after the text appeared.

To modify or retrieve the value of any of the `TE_SILENT` system variables, use the `TE_getvar` and `TE_setvar` functions. For more information, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the Vuser types in its name, and then waits for the application to respond.

```
/* Declare variables for TE_wait_text */
int ret_row;
int ret_col;
char ret_text [80];

/* Type in user name. */
TE_type ("John");

/* Wait for teller to respond. */
TE_wait_text ("Enter secret code:", 30, 29, 13, 1, 13, &ret_col, &ret_row,
             ret_text);
```

You can instruct VuGen to automatically generate `TE_wait_text` statements, and insert them into a script, while you record the script.

To instruct VuGen to automatically generate `TE_wait_text` statements, and insert them into a script while recording:

- 1** Select **Vuser > Recording Options**. The Recording Settings dialog box appears.
- 2** Under **Generate Automatic Synchronization Commands**, select the **Prompt** check box, and then click **OK**.

The following is an example of a `TE_wait_text` statement that was automatically generated by VuGen. The function waits up to 20 seconds for the string "keys" to appear anywhere on the screen. Note that VuGen omits all the optional parameters when it generates a `TE_wait_text` function.

```
TE_wait_text("keys", 20);
```

Waiting for the Terminal to be Silent

In instances when neither cursor synchronization nor text synchronization are effective, you can use "silent synchronization" to synchronize the script. With "silent synchronization," the Vuser waits for the terminal emulator to be silent for a specified period of time. The emulator is considered to be silent when it does not receive any input from the server for a specified period of time.

Note: Use silent synchronization only when neither cursor synchronization nor text synchronization are effective.

You use the `TE_wait_silent` function to instruct a script to wait for the terminal to be silent. You specify the period for which the terminal must be silent. If the terminal is silent for the specified period, then the `TE_wait_silent` function assumes that the application has stopped printing text to the terminal screen, and that the screen has stabilized.

The syntax of the function is:

```
int TE_wait_silent (int sec, int milli, int timeout);
```

The **TE_wait_silent** function waits for the terminal emulator to be silent for the time specified by *sec* (seconds) and *milli* (milliseconds). The emulator is considered silent when it does not receive any input from the server. If the emulator does not become silent (i.e. stop receiving characters) during the time specified by the time *timeout* variable, then the function returns an error.

For example, the following statement waits for the screen to be stable for three seconds. If after ten seconds, the screen has not become stable, the function returns an error.

```
TE_wait_silent (3, 0, 10);
```

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

54

RTE - Reading Text from Terminal Screen

RTE Vusers can read text from the user interface of a terminal emulator, and then perform various tasks with that text.

This chapter includes:

- ▶ About Reading Text from the Terminal Screen on page 791
- ▶ Searching for Text on the Screen on page 792
- ▶ Reading Text from the Screen on page 792

About Reading Text from the Terminal Screen

There are several Vuser functions that RTE Vusers can use to read text from the terminal screen. You can use these functions, **TE_find_text** and **TE_get_text_line**, to check that the terminal emulator is responding correctly, or to enhance the logic in your scripts.

After recording, you can manually insert **TE_find_text** and **TE_get_text_line** statements directly into your RTE Vuser scripts.

Searching for Text on the Screen

The `TE_find_text` function searches for a line of text on the screen. The syntax of the function is:

```
int TE_find_text (char *pattern, int col1, int row1, int col2, int row2,
                 int *retcol, int *retrow, char *match);
```

This function searches for text matching *pattern* within the rectangle defined by *col1*, *row1*, *col2*, *row2*. Text matching the pattern is returned to *match*, and the actual row and column position is returned to *retcol* and *retrow*. The search begins in the top-left corner. If more than one string matches *pattern*, the one closest to the top-left corner is returned.

The **pattern** can include a regular expression. See the *Online Function Reference* for details on using regular expressions.

You must manually type `TE_find_text` statements into your Vuser scripts. For details on the syntax of the `TE_find_text` function, see the *Online Function Reference* (**Help > Function Reference**).

Reading Text from the Screen

The `TE_get_text_line` function reads a line of text from the area of the screen that you designate. The syntax of the function is:

```
char *TE_get_text_line (int col, int row, int width, char *text);
```

This function copies a line of text from the terminal screen to a buffer *text*. The first character in the line is defined by *col*, *row*. The column coordinate of the last character in the line is indicated by *width*. The text from the screen is returned to the buffer *text*. If the line contains tabs or spaces, the equivalent number of spaces is returned.

In addition, the **TE_get_cursor_position** function can be used to retrieve the current position of the cursor on the terminal screen. The **TE_get_line_attribute** function returns the character formatting (for instance, bold or underline) of a line of text.

You must manually type **TE_get_text_line** statements into your Vuser scripts. For details on the syntax of the **TE_get_text_line** function, see the *Online Function Reference* (**Help > Function Reference**).

55

Mailing Services Protocols

VuGen allows you to test several mailing services on a protocol level. It emulates the sending of mail, and most of the standard operations performed against a mail server.

This chapter includes:

- About Developing Vuser Scripts for Mailing Services on page 795
- Getting Started with Mailing Services Vuser Scripts on page 796
- Understanding IMAP Scripts on page 797
- Understanding MAPI Scripts on page 798
- Understanding POP3 Scripts on page 800
- Understanding SMTP Scripts on page 801

About Developing Vuser Scripts for Mailing Services

The Mailing Service protocols emulate a user working with an email client, viewing and sending emails. The following mailing services are supported:

- Internet Messaging (IMAP)
- MS Exchange (MAPI)
- Post Office Protocol (POP3)
- Simple Mail Transfer Protocol (SMTP)

The mail protocols support both record and replay, with the exception of MAPI that only supports replay.

When you record an application using one of the mail protocols, VuGen generates functions that emulate the mail client's actions. You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see "Setting Script Generation Preferences" in *Volume I-Using VuGen*. If the communication is performed through multiple protocols, you can record both of them. You can record several mail protocols, or a mail protocol together with HTTP or WinSock. For instructions on specifying multiple protocols, see "Recording with VuGen" in *Volume I-Using VuGen*.

All Mailing Service functions come in pairs—one for global sessions and one where you can indicate a specific mail session. For example, **imap_logon** logs on to the IMAP server globally, while **imap_logon_ex** logs on to the IMAP server for a specific session.

Getting Started with Mailing Services Vuser Scripts

This section provides an overview of the process of developing Vuser scripts for Mailing Services using VuGen.

To develop a Mailing Service Vuser script:

1 Create a basic script using VuGen.

Invoke VuGen and create a new Vuser script for either a single mail protocol or multiple protocols.

2 Record the basic script using VuGen. (Except MAPI)

Select an application to record. Perform typical operations in your application. For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

For MAPI, recording is not supported. Instead, you create an empty MAPI script and manually insert **mapi** functions into it. For examples, see the *Online Function Reference* (**Help > Function Reference**).

3 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see "Correlating Statements" in *Volume I-Using VuGen*.

6 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

7 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding IMAP Scripts

IMAP Vuser script functions record the Internet Mail Application Protocol. Each IMAP function begins with an **imap** prefix.

For detailed syntax information on these functions, see the *Online Function Reference (Help > Function Reference)*.

In the following example, the **imap_create** function creates several new mailboxes: Products, Solutions, and FAQs.

```
Actions()
{
    imap_logon("ImapLogon",
              "URL=imap://johnd:letmein@exchange.mycompany.com",
              LAST);

    imap_create("CreateMailboxes",
               "Mailbox=Products",
               "Mailbox=Solutions",
               "Mailbox=FAQs",
               LAST);

    imap_logout();

    return 1;
}
```

Understanding MAPI Scripts

MAPI Vuser script functions record activity to and from an MS Exchange server. Each MAPI function begins with a **mapi** prefix.

Note: To run MAPI scripts, you must define a mail profile on the machine running the script. For example, install Outlook Express, set it as the default mail client, and create a mail account. Alternatively, install Microsoft Outlook, set it as the default mail client, create a mail account and create a mail profile. To create a mail profile in Microsoft Outlook, select **Settings > Control Panel > Mail > Show Profiles** and add a mail profile.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the `mapi_send_mail` function sends a sticky note through an MS Exchange server.

```
Actions()
{
    mapi_logon("Logon",
              "ProfileName=John Smith",
              "ProfilePass=Tiger",
              LAST);
    //Send a Sticky Note message
    mapi_send_mail("SendMail",
                  "To=user1@techno.merc-int.com",
                  "Cc=user0002t@techno.merc-int.com",
                  "Subject=<GROUP>:<VUID> @ <DATE>",
                  "Type=lpm.StickyNote",
                  "Body=Please update your profile today.",
                  LAST);

    mapi_logout();
    return 1;
}
```

Understanding POP3 Scripts

POP3 Vuser script functions emulate actions using the Post Office Protocol, POP3. Each function begins with a **pop3** prefix.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the **pop3_retrieve** function retrieves five messages from the POP3 server.

```

Actions()
{
  pop3_logon("Login", "
    URL=pop3://user0004t:my_pwd@techno.merc-int.com",
    LAST);

  // List all messages on the server and receive that value
  totalMessages = pop3_list("POP3", LAST);

  // Display the received value (It is also displayed by the pop3_list function)
  lr_log_message("There are %d messages.\r\n\r\n", totalMessages);

  // Retrieve 5 messages on the server without deleting them
  pop3_retrieve("POP3", "RetrieveList=1:5", "DeleteMail=false", LAST);
  pop3_logoff();
  return 1;
}

```


Understanding SMTP Scripts

SMTP Vuser script functions emulate the Single Mail Transfer Protocol traffic. Each SMTP function begins with an **smtp** prefix.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the **smtp_send_mail** function sends a mail message, through the SMTP mail server, techno.

```

Actions()
{
    smtp_logon("Logon",
        "URL=smtp://user0001t@techno.merc-int.com",
        "CommonName=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;
}

```


56

Tuxedo Protocols

You use VuGen to record communication between a Tuxedo client application and a Tuxedo application server. The resulting script is called a Tuxedo Vuser script.

This chapter includes:

- About Tuxedo Vuser Scripts on page 804
- Getting Started with Tuxedo Vuser Scripts on page 805
- Understanding Tuxedo Vuser Scripts on page 806
- Viewing Tuxedo Buffer Data on page 809
- Defining Environment Settings for Tuxedo Vusers on page 810
- Debugging Tuxedo Applications on page 811
- Correlating Tuxedo Scripts on page 811

About Tuxedo Vuser Scripts

When you record a Tuxedo application, VuGen generates LRT functions that describe the recorded actions. These functions emulate communication between a Tuxedo client and a server. Each LRT function begins with an **lrt** prefix.

In addition to the **lrt** prefix, certain functions use an additional prefix of **tp**, **tx** or **F**. These sub-prefixes indicate the function type, similar to the actual Tuxedo functions. The **tp** sub-prefix indicates a Tuxedo client tp session. For example, **lrt_tpcall** sends a service request and awaits its reply. The **tx** sub-prefix indicates a global tx session. For example, **lrt_tx_begin** begins a global transaction. The **F** sub-prefix indicates an FML buffer related function. For example, **lrt_Finitialize** initializes an existing buffer.

Functions without an additional prefix emulate standard C functions. For example, **lrt_strcpy** copies a string, similar to the C function **strcpy**.

You can view and edit the recorded script from VuGen's main window. The LRT functions that are recorded during the session are displayed in the VuGen window, allowing you to visually track your network activities.

Before You Record

Before you record, verify that the Tuxedo directory, %TUXDIR%\bin is in the path.

If the environment variables have changed since the last time you restarted VuGen, VuGen may record the original variable value rather than the current value.

To avoid any inconsistencies, you should restart VuGen before recording Tuxedo applications.

Getting Started with Tuxedo Vuser Scripts

This section provides an overview of the process of developing Tuxedo Vuser scripts using VuGen.

To develop a Tuxedo Vuser script:

1 Record the basic script using VuGen.

Invoke VuGen and create a new Vuser script. Specify **Tuxedo 6** (for recording Tuxedo Version 6.x) or **Tuxedo** (for recording Tuxedo Version 7.x) as the type of Vuser. Select an application to record. Record typical operations on your application.

For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see "Correlating Statements" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Understanding Tuxedo Vuser Scripts

VuGen records typical Tuxedo sessions and generates Tuxedo-specific functions. The functions use an **lrt** prefix. For example, **lrt_tpcall** sends a service request.

The LRT functions are divided into the following categories: Buffer Manipulating, Client/Server Session, Communication, Environment Variable, Error Processing, Transaction Handling, and Correlation functions.

You can also manually program any of the functions into your script. For syntax and examples of the LRT functions, see the *Online Function Reference (Help > Function Reference)*.

After you record a session, VuGen's built-in editor lets you view the recorded code. You can scroll through the script, see Tuxedo statements that were generated by your application, and examine the data that was returned by the server. The VuGen window provides you with valuable information about the recorded Tuxedo session. When you view the script in the main window, you see the sequence in which VuGen recorded your activities.

In the following example, VuGen recorded a client's actions in a Tuxedo bank application. The client performed an action of opening a bank account and specifying all the necessary details. The session was aborted when the client specified a zero opening balance.

```
lrt_abort_on_error();  
lr_think_time(65);  
tpresult_int = lrt_tpbegin(30, 0);  
data_0 = lrt_tpalloc("FML", "", 512);  
lrt_Finitialize((FBFR*)data_0);
```

```

/* Fill the data buffer data_0 with new account information */
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=8",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=C",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=MID_INIT", "value=Q", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=PHONE", "value=123-456-7890",
LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
New York, NY 10000", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=SSN", "value=111111111", LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=LAST_NAME",
"value=Doe", LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=FIRST_NAME",
"value=BJ", LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT",
"value=0.00", LRT_END_OF_PARMS);

/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, &data_0, &olen_2, 0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();

```

Using Parameters in Tuxedo Scripts

You can define parameters in Tuxedo scripts, as described in "Creating Parameters" in *Volume I-Using VuGen*. Note that Tuxedo scripts contain strings of type "name=..." or "value=...". You can only define parameters for the portion of the string following the equal sign (=). For example:

```

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

```

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

Running Tuxedo Scripts

If you encounter problems recording or running Tuxedo applications, check that the Tuxedo application runs without VuGen, and that the environment variables have been defined correctly. For more information, see "Viewing Tuxedo Buffer Data" below. Note that after you set or modify the Tuxedo variables, you should restart VuGen and your application, in order for the changes to take effect. If your application is 16-bit, then you also need to kill the NTVDM process.

If you experience problems during execution, check the Tuxedo log file on the side of the server for error messages. By default, this file is found in the directory indicated by the environment variable APPDIR. The file name has the form ULOG.mmddyy, where mmddyy indicates the current month, day, and year. The file for March 12, 1999 would be ULOG.031299. The default location of this file can be changed by setting the environment variable ULOGPFX on the server. A log file can also be found on the client side, in the current directory, unless the ULOGPFX variable changes its location.

Viewing Tuxedo Buffer Data

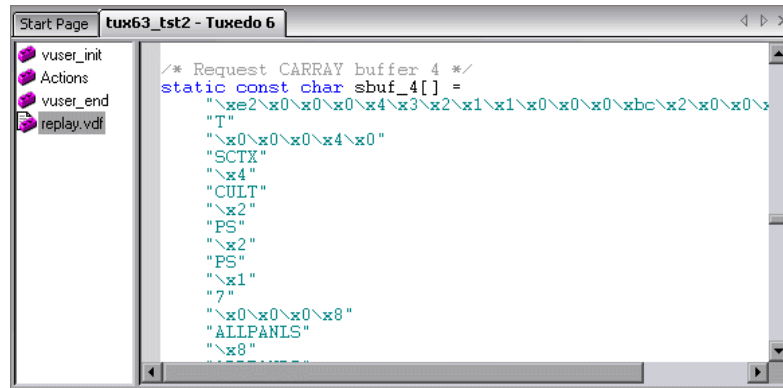
When you use VuGen to create a Tuxedo Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**.

The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file is called **replay.vdf**, and it contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRT functions use the buffer descriptors to access the data.

You can use VuGen to view the contents of the data file by selecting the **replay.vdf** file in the left pane's tree view.

The option to view a data file is available by default for Tuxedo scripts.



```

Start Page tux63_tst2 - Tuxedo 6
vuser_init
Actions
vuser_end
replay.vdf
/* Request CARRAY buffer 4 */
static const char sbuf_4[] =
"\xe2\x0\x0\x0\x4\x3\x2\x1\x1\x0\x0\x0\xbc\x2\x0\x0\x
"T
"\x0\x0\x0\x4\x0"
"SCTX"
"\x4"
"CULT"
"\x2"
"PS"
"\x2"
"PS"
"\x1"
"7"
"\x0\x0\x0\x8"
"ALLPANLS"
"\x8"

```

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)
```

Debugging Tuxedo Applications

In general, use **Tuxedo 6** to record applications using Tuxedo 6.x or earlier, and use **Tuxedo** to record applications using Tuxedo 7.1 and higher.

If you encounter problems recording or replaying Tuxedo applications, or the script is missing a call to `lrt_tpinitialize`, contact Customer Support to check which DLLs are used with the application.

If the application uses **wtuxws32.dll**, instead of **libwsc.dll**, contact Customer Support to obtain a patch to enable the recording.

Correlating Tuxedo Scripts

VuGen supports correlation for Vuser scripts recorded with Tuxedo applications. Correlated statements enable you to link statements by saving a portion of a buffer and use it in subsequent statements.

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

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

For additional information about the syntax of these functions, see the *Online Function Reference*.

Correlating FML and FML32 Buffers

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

To correlate statements using `lrt_save_fld_val`:

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

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

- 2 Reference the parameter.

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

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

```
/* Fill the data_0 buffer with new account information*/
data_0 = lrt_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_fld((FBFR*)data_0, "name=ACCOUNT_ID", "value={account_id}",
LRT_END_OF_PARMS);
lrt_fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=200.11", ...);
```

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

You can correlate by field ID as follows:

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

Correlating Character Strings

Use `lrt_save_parm` or `lrt_save_searched_string` to correlate character strings.

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

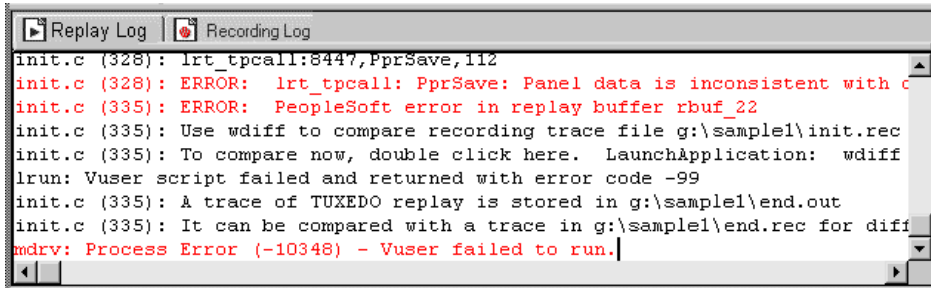
Determining Which Values to Correlate

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

To compare the log files:

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

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



```

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. LaunchApplication: 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.

WDiff opens and the differences between the record and replay files are highlighted in yellow. For more details about the Wdiff utility, see "Correlating Statements" in *Volume I-Using VuGen*.

To correlate statements using lrt_save_parm:

Once you decide which value to correlate, you can use **lrt_save_parm** to save a portion of a character array (such as a STRING or CARRAY buffer) to a parameter.

- 1 Insert the **lrt_save_parm** statement in your script at the point where you want to save the contents of the current buffer.

lrt_save_parm (buffer, offset, length, "**param_name**");

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

View the buffer contents by selecting the **replay.vdf** file in the Data Files box of the main VuGen window.

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

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

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

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

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

To correlate statements using `lrt_save_searched_string`:

Use `lrt_save_searched_string` to search for a string in a buffer, and save a portion of the buffer, relative to the string occurrence, to a parameter.

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

```
lrt_save_searched_string (buffer, buf_size, occurrence, string, offset,
                        length, "param_name");
```

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

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

View the buffer contents by selecting the `replay.vdf` file in the Data Files box of the main VuGen window.

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

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

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

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

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

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

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

Real and Media Player Protocols

Streaming media is a rapidly growing market that allows for the delivery of audio/visual content over the Internet. The idea behind streaming media is that the audio/video content can be transmitted to the end user without having to first download the file in its entirety. Streaming works by having the server continuously stream the content to the client as it displays it.

RealPlayer is an application that display streaming content.

You use VuGen to record communication between a client application and a server that communicate using the RealPlayer protocol. The resulting script is called a Real Vuser script.

This chapter includes:

- About Recording Streaming Data Virtual User Scripts on page 820
- Getting Started with Streaming Data Vuser Scripts on page 820
- Using RealPlayer LREAL Functions on page 821
- Using Media Player MMS Functions on page 822

Note: The Media Player (MMS) protocol should not be confused with the Multimedia Messaging Service (MMS) protocol. For information, see "MMS (Multimedia Messaging Service) Vuser Scripts" on page 831

About Recording Streaming Data Virtual User Scripts

The Streaming Data protocols allows you to emulate a user playing media or streaming data files.

When you record an application using a streaming data protocol, VuGen generates functions that describe your actions. For RealPlayer sessions, VuGen generates functions with an **ireal** prefix. For Media Player sessions, VuGen uses functions with an **mms** prefix. Note that recording is not supported for Media Player mms functions—only replay.

Getting Started with Streaming Data Vuser Scripts

This section provides an overview of the process of developing RealPlayer and Media Player streaming data Vuser scripts using VuGen.

To develop a Real or Media Player Vuser script:

1 Record the basic script using VuGen. (Real only)

Invoke VuGen and create a new Vuser script. Select an application to record, and record typical operations on your application. For details, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see "Creating Parameters" in *Volume I-Using VuGen*.

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see "Correlating Statements" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using RealPlayer LREAL Functions

The functions developed to emulate communication between a client and a server by using the RealPlayer protocol are called Real Player functions. Each Real Player function has an **lreal** prefix.

VuGen automatically records most of the LREAL functions listed in this section during a Real Player session. You can also manually program any of the functions into your script.

For example, the **lreal_play** function takes the form:

```
int lreal_play (int miplayerID, long mulTimeToPlay);
```

To play the clip until the end, use any negative value for **mulTimeToPlay**. To play the clip for a specific duration number of milliseconds, specify the number of milliseconds. **miplayerID** represents a unique ID of a RealPlayer instance.

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

Using Media Player MMS Functions

The functions developed to emulate client/server communication for Media Player's MMS protocol, are called MMS Virtual User functions—each function has an **mms** prefix.

Important: In order to replay Media Player functions, you must place a file called **wmload.asf** on the Windows Media server machine. The VuGen machine must be able to access using **mms://<servername>/wmload.asf**. This ASF file can be any media file renamed to **wmload.asf**.

All MMS functions come in pairs—one for global sessions and one for a specific session. For example, **mms_close** closes the Media Player globally, while **mms_close_ex** closes the Media Player for a specific session.

A typical function, **mms_play**, takes the following form:

```
int mms_play (char message, <List of Attributes>, LAST);
```

In the following example, the **mms_play** function plays an **asf** file for different durations:

```
//Play for a duration of 10 seconds.
mms_play("Welcome","URL=mms://server/welcome.asf",
duration=10",
LAST);

//Play the clip until its completion, after waiting 5 seconds.
mms_play ("Welcome", "URL=mms://server/welcome.asf",
"duration=-1",
"starttime=5",
LAST);
```

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

58

Wireless Protocols

VuGen enables you to generate Wireless Vuser scripts by recording typical Wireless sessions. When you run a script, the resulting Vuser emulates activity between your toolkit or phone and Web server (or gateway for WAP).

This chapter includes:

- ▶ Understanding the WAP Protocol on page 823
- ▶ Getting Started with Wireless Vuser Scripts on page 825
- ▶ Using Wireless Vuser Functions on page 827
- ▶ Push Support on page 828
- ▶ VuGen Push Support on page 830
- ▶ MMS (Multimedia Messaging Service) Vuser Scripts on page 831
- ▶ Running an MMS Scenario in the Controller on page 832

Understanding the WAP Protocol

The Wireless Application Protocol (WAP) is an open, global specification that enables mobile users with wireless devices to instantly access and interact with information and services.

The WAP protocol specifies a microbrowser thin-client using a new standard called WML that is optimized for wireless handheld mobile terminals. WML is a stripped-down version of XML.

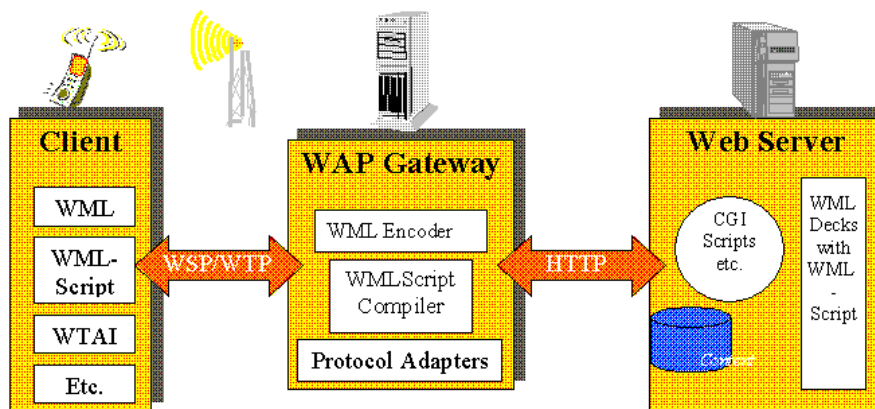
WAP also specifies a proxy server that:

- ▶ acts as a gateway between the wireless network and the wire-line Internet
- ▶ provides protocol translation
- ▶ optimizes data transfer for the wireless handset

WAP architecture closely resembles the WWW model. All content is specified in formats that are similar to the standard Internet formats. Content is transported using standard protocols in the WWW domain and an optimized HTTP-like protocol in the wireless domain (Wireless Session Protocol). You locate all WAP content using WWW standard URLs.

WAP uses many WWW standards, including authoring and publishing methods. WAP enhances some of the WWW standards in ways that reflect the device and network characteristics. WAP extensions are added to support Mobile Network Services such as Call Control and Messaging. It accounts for the memory and CPU processing constraints that are found in mobile terminals. WAP also supports low bandwidth and high latency networks.

WAP assumes the existence of a gateway that is responsible for encoding and decoding data transferred to and from the mobile client. The purpose of encoding content delivered to the client is to minimize the size of data sent to the client over-the-air, as well as to minimize the computational energy required by the client to process that data. The gateway functionality can be added to origin servers, or placed in dedicated gateways as illustrated below.



WAP Toolkits

To assist developers in producing WAP applications and services, the leading companies such as Nokia, Ericsson, and Phone.com, have developed toolkits. The WAP Toolkit provides an environment for developers who want to provide Internet services and content for mobile terminals. It allows developers to write, test, debug, and run applications on a PC-based simulator phone. The toolkit allows users to browse WAP sites through an HTTP connection or a WAP gateway.

A mobile phone communicates with a gateway in WSP protocol; a toolkit can communicate with the gateway, or directly with the server. VuGen automatically detects the communication mode that is configured in the toolkit: WSP or HTTP. If you are interested in the traffic to the gateway, you record in WSP mode. If you want to check the server and the content providers, you can record your toolkit session in HTTP mode, and bypass the gateway.

VuGen uses custom API functions to emulate a user session. Most functions are the standard Web protocol functions utilizing the HTTP protocol. Several WAP functions emulate actions specific to WAP Vusers. For a list of the supported functions, see "Using Wireless Vuser Functions" on page 827.

Getting Started with Wireless Vuser Scripts

Wireless Vuser Scripts emulate a user using a wireless browser. You record the user browsing on a PC-based simulator phone (toolkit). You can then distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

This section provides an overview of the process of developing Wireless Vuser scripts using VuGen.

To develop a Wireless script:

1 Create a new script using VuGen.



Select **File > New** or click the **New** button to create a new script in either single or multiple protocol mode.

For details about creating a new script, see "Recording with VuGen" in *Volume I-Using VuGen*.

2 Record the actions using VuGen.

Record the actions over the toolkit session. VuGen automatically detects the toolkit settings and uses those settings during recording.

For information about recording, see "Recording with VuGen" in *Volume I-Using VuGen*.

3 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see "Enhancing Vuser Scripts" in *Volume I-Using VuGen*.

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values. For details, see "Creating Parameters" in *Volume I-Using VuGen*.

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include the run logic, pacing, logging, think time, performance preferences, and gateway settings.

For information about the General run-time settings, see "Configuring Run-Time Settings" in *Volume I-Using VuGen*.

For information about common Internet protocol run-time settings, see Chapter 23, "Configuring Network Run-Time Settings" in *Volume I-Using VuGen*.

For information about WAP specific run-time settings, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

6 Perform correlation.

Check your script to determine if there are dynamic values that require correlation. For Wireless protocols, you perform manual correlation by adding **web_reg_save_param** functions.

For more information, see "Performing Manual Correlation" on page 655.

7 Save and run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly. While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script.

For details about running the Vuser script as a standalone test, see "Running Vuser Scripts in Standalone Mode" in *Volume I-Using VuGen*.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Using Wireless Vuser Functions

The functions developed to emulate communication between a wireless instrument and Web server (or gateway for WAP), are called Vuser functions. Some functions are generated when you record a script; others you must manually insert into the script. You can also add Vuser message functions and custom C functions to your Vuser scripts after recording.

General Vuser functions begin with an **lr** prefix. The functions representing standard HTTP actions, have a **web** prefix.

Functions that are specific to WAP, have a **wap** prefix. For example, **wap_connect** connects to a WAP gateway. WAP functions are divided into the following categories: Bearer, Connection, Formatting, Gateway, Push, and Radius functions.

Functions that emulate a RAS or NAS server during a WAP session, have a **radius** prefix. For example, **radius_account** authenticates a user to a RADIUS server.

For a complete list of all VuGen functions, see the *Online Function Reference* (**Help > Function Reference**).

For WAP Vusers running scripts in Wireless Session Protocol (WSP) mode, only the following functions are supported:

Action Functions	<code>web_custom_request</code> , <code>web_submit_data</code> , and <code>web_url</code>
Authentication Functions	All— <code>web_set_user</code> , <code>web_set_certificate[_ex]</code>
Cookie Functions	All— <code>web_add_cookie</code> , <code>web_cleanup_cookie</code> , <code>web_remove_cookie</code>
Header Functions	All — <code>web_add_auto_header</code> , <code>web_add_header</code> , <code>web_cleanup_auto_headers</code> , <code>web_save_header</code>
Correlation Functions	All— <code>web_create_html_param[_ex]</code> , <code>web_reg_save_param</code> , <code>web_set_max_html_param_len</code>

Push Support

In the normal client/server model, a client requests information or a service from a server. The server responds by transmitting information or performing a service to the client. This is known as **pull** technology—the client pulls information from the server.

In contrast to this, there is also **push** technology. The WAP push framework transmits information to a device without a previous user action. This technology is also based on the client/server model, but there is no explicit request from the client before the server transmits its content.

To perform a push operation in WAP, a **Push Initiator** (PI) transmits content to a client. However, the Push Initiator protocol is not fully compatible with the WAP Client—the Push Initiator is on the Internet, and the WAP Client is in the WAP domain. Therefore, we need to insert a translating gateway to serve as an intermediary between the Push Initiator and the WAP Client. The translating gateway is known as the **Push Proxy Gateway** (PPG).

The access protocol on the Internet side is called the **Push Access Protocol** (PAP).

The protocol on the WAP end is called the Push **Over-The-Air** (OTA) protocol.

The Push Initiator contacts the Push Proxy Gateway (PPG) over the Internet using the PAP Internet protocol. PAP uses XML messages that may be tunneled through various well-known Internet protocols such as HTTP. The PPG forwards the pushed content to the WAP domain. The content is then transmitted using the OTA protocol over the mobile network to the destination client. The OTA protocol is based on WSP services.

In addition to providing basic proxy gateway services, the PPG is capable of notifying the Push Initiator about the final status of the push operation. In two-way mobile networks, it can also wait for the client to accept or reject the content.

Push Services Types

Push services can be of the SL or SI type:

- ▶ **SL.** The Service Loading (SL) content type provides the ability to cause a user agent on a mobile client to load and execute a service—for example, a WML deck. The SL contains a URI indicating the service to be loaded by the user agent without user intervention when appropriate.
- ▶ **SI.** The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. For example, the notifications may be about new emails, changes in stock price, news headlines, and advertising.

In its most basic form, an SI contains a short message and a URI indicating a service. The message is presented to the end-user upon reception, and the user is given the choice to either start the service indicated by the URI immediately, or postpone the SI for later handling. If the SI is postponed, the client stores it and the end-user is given the ability to act upon it at a later point of time.

VuGen Push Support

Push support for VuGen is divided into three parts:

- Push support at the client end—the ability to accept push messages.
- Push support to WAP HTTP Vusers—emulating Push Initiators.
- Push messages (SI & SL) format services—formatting push messages.

Client Push Support

At the client end, VuGen supports both push services (SL and SI) for all replay modes (CO and CL). The `wap_wait_for_push` function instructs the Vuser to wait for a push message to arrive. You set the timeout for this function in the run-time settings.

When a push message arrives, the Vuser parses it to determine its type and to retrieve its attributes. If parsing was successful, it generates and executes a pull transaction to retrieve the relevant data. You can disable the pull event, indicating to the Vuser not to retrieve the message data by configuring the Run-Time settings. For more information, see Chapter 24, "Run-Time Settings for Selected Protocols" in *Volume I-Using VuGen*.

Emulating a Push Initiator

Push support for WAP HTTP Vusers enables you to perform load testing of the PPG. Push support allows Vusers to function as Push Initiators supporting the **Push Access Protocol (PAP)**. The PAP defines the following sets of operations between the PI and the PPG:

- Submit a Push request.
- Cancel a Push request.
- Submit a query for the status of a push request.
- Submit a query for the status of a wireless device's capabilities.
- Initiate a result notification message from the PPG to the PI.

All operations are request/response—for every initiated message, a response is issued back to the PI. PI operations are based on the regular HTTP POST method supported by VuGen. Currently, only the first two operations are supported through **wap_push_submit** and **wap_push_cancel**.

You can submit data to a Web server using the **web_submit_data** function. It is difficult, however, to send long and complex data structures using this function. To overcome this difficulty and provide a more intuitive API function, several new API functions were added to properly format the XML message data: **wap_format_si_msg** and **wap_format_sl_msg**. For more information about these functions, see the *Online Function Reference*.

MMS (Multimedia Messaging Service) Vuser Scripts

MMS (Multimedia Messaging Service) is an extension of the SMS protocol. Whereas SMS messages can only contain text, MMS allows you to send and receive messages with a wide range of content to MMS capable handsets. This content can be in the form of text, sound, email messages, images, video clips, and even streaming data. It is also possible to send multimedia messages from a mobile phone to an email address.

An MMS message typically includes a collection of attachments. While SMS messages are limited to 160 bytes, an MMS message could be several MBs in size. MMS usually requires a third generation (3G) network to enable such large messages to be delivered.

To receive an MMS message, a mobile phone receives an MMS notification over SMS. The SMS message can be received over various SMS protocols such as SMPP, UCP, and CIMD2. The SMS message contains a unique path to the MMS message stored in the MMSC server's database and the mobile phone uses this path to download the message from the SMSC. The current version of VuGen supports the receiving of MMS notifications over the SMPP interface.

Multimedia Messaging Service Vuser scripts support the 1.0 and 1.1 versions of the MMS protocol, as defined by OMA (Open Mobile Alliance organization). Using MMS Vusers, you can send MMS messages to the MMSC server directly over the HTTP protocol, or over the WAP protocol through a WAP gateway.

Multimedia Messaging Service functions emulate the sending and receiving of MMS messages. Each function begins with an **mm** prefix. For detailed syntax information for these functions, see the *Online Function Reference* (**Help > Function Reference**).

Running an MMS Scenario in the Controller

An MMS (Multimedia Messaging Service) scenario requires a command line setting.

To set the MMS command line setting:

- 1** From the Scenario Schedule screen, click **Details**. The Group Information dialog is displayed.
- 2** If the Command line box is not visible, click the **More** button.
- 3** Add the following to the end of the Command line text: `-usingwininet yes`
- 4** Click **OK** to accept the Command line switch.

Index

Numerics

-25210 423

A

abstract types 103

Action

method (Java) 430

steps in Web scripts 609

Actions class 429

Add Rule 651

agent for citrix presentation server 325

Agent for Microsoft Terminal Server

tips 353

agent for microsoft terminal server 351

AJAX (Click and Script) Vuser Scripts

about 473

recording 474

AJAX controls 473

algorithm, for encryption 181

ALNUM flag 599

AMF Call Properties dialog box 486

AMF terms 479

AMF Vuser scripts

about 477

call properties 486

correlating 481

envelope header set properties 488

functions 480

header set properties 487

recording scripts 477

understanding 485

viewing 485

ANSI C support, in custom scripts 421

Any type

manipulating 35

XSD, running 130

append snapshot 345

Application Deployment Solution, Citrix

Vuser type 299–323

application server, Oracle NCA 674

arrays

Web Services arguments 101

Web Services duplicating 33

Web Services excluding elements 32

arrays, XML 101

assemblies, adding in .NET 543

assertion, SAML 152

AssignToParam property (Web) 603

asynchronous messages 202

attachments

MIME, .NET toolkit 222

attachments, WSDL 113

authentication

username (message) 174

username (transport) 174

authentication for WSDLs 43

authentication mode

for custom binding 176

automation compliant 418

B

base 64 encoding 108

BIN flag 599

binary coded data 606

binary encoding 180

binary view of data (WinSock) 390

bitmap mismatch 315, 344

bookmarks

in data (WinSock) 393

Bootstrap policy 175

boundaries, defining for correlation 660

buffer capacity, increasing (WS) 185

Index

buffer navigator (WinSock) 391
BytesMessage 201

C

C language support

conventions 421

C Vusers 419

cache

loading and dumping 585

capture file, generating 80

CARRAY buffers 813

Certificate Authentication 173

certificates

selecting for Web Services 177

SSL for server traffic 86

Check Properties dialog box 630

checkpoints

setting 122

Web Services scripts 122

checks (Web)

defining properties for 603

image checks 600

modifying in scripts 630

overview 589

text 592

types of 591

choice elements 99

choice optional elements 107

choose iteration, Web Services 92

citrix

agent for citrix presentation server
325

Citrix agent 325

Citrix server, disconnecting 303

Citrix Vuser scripts 299–323

client version 302

disconnecting from server 303

display settings 306

editing 307

function 317

getting started 301

synchronizing replay 308

tips for record and replay 318

client for Citrix 302

clientVia behavior 186

clipboard, in RDP 339

COM

data types 446

overview and interfaces 444

COM Vuser scripts

class context 445

creating object instances 463

developing 443

error checking 462

getting started 446

IDispatch interface 466

instantiating objects 463

interface pointers 460

log files for debugging 447

recording options 450

retrieving an interface 464

scanning for correlations 468

script structure 460

selecting COM objects to record 447

type libraries 445

understanding 460

command line arguments

reading in Java Vuser scripts 438

comments

adding to correlation steps 638

comparing

XML files 56

comparison method

HTML vs. text 649

comparison options, WSDL/XML 52

comparison reports 54

configuration files

SAML security 223

user handler, mmdrv 223

configuring

application security and permissions
519

Connect dialog box (RTE) 769

connection settings 43

connections, closing open ones (.NET) 510

Control steps, modifying (Web) 626

converting

custom request to C 623

Web functions to Java 560

coordinate shifting (RDP) 350

copy and paste

- advanced for WinSock Vusers 396
- RTE Vusers 771
- Correlated Query tab
 - COM 469
 - Database 374
- correlating
 - advanced properties 641
 - after recording (Web, Wireless) 643
 - built-in detection 634
 - COM Vusers 468
 - for known contexts (Web) 634
 - HTML statements (Web) 631
 - Java statements 267
 - maximum parameter size 635
 - Microsoft .NET scripts 516
 - rules for Web Vusers 635
 - scanning Database Vuser script 374
 - Siebel-Web 743
 - SWECOUNT 751
 - Tuxedo 816
 - with snapshots (Web) 643
- Correlation Results tab 651
- CtLib
 - result set errors 371
- custom binding, Web Service security 176
- Custom Request dialog box (Web) 624
- Custom Request step
 - defined 581
 - for XML 667
 - modifying (Web) 623
- Custom Vuser types
 - C Vusers 419
 - Java Vusers 422
 - JavaScript Vusers 426
 - VB Vusers 423
 - VBScript Vusers 425

D

- data buffers
 - Tuxedo Vuser scripts 809
 - WinSock Vuser scripts 402
- data files
 - Windows Sockets Vuser scripts 403
- data grids
 - viewing, .NET 515

- Database Vuser scripts
 - correlating 373
 - developing 358
 - getting started 361
 - handling errors 370
 - return codes 369
 - row information 364
 - using lrd functions 362
 - viewing grids 366
- dataset action, Web Services 142
- DB2-CLI 357
- DbLib 357
- DCOM tab 452
- decode to file 112
- deep correlation (Java) 269
- defining properties, text checks 592
- deleting steps
 - from Web scripts 608
- derived types
 - multiple roots 37
 - setting arguments 103
- detector, EJB 280
- DIG flag 599
- digital signatures 150
- disabling functions (SAPGUI) 722
- distinguished names 506
- DN (LDAP) 506
- DNS Vusers
 - functions 380
 - overview 379
- duplex communication 523
- duplicating arrays 33
- dynamic ports 412

E

- EBCDIC translation 405
- editor for XML 29
- EJB
 - instance 291
 - method 293
 - Vuser scripts 279
- EJB Detector
 - about 290
 - command-line 282
 - log files 284

Index

- setup 280
- element types, choice 99
- encoding, for WS config. file 180
- encrypted data for Web Services security 150
- end method 430
- entropy mode 181
- environment settings
 - Java 439
 - Tuxedo Vusers 810
- Ericom 761
- Error
 - 25210 423
- error handling
 - COM Vuser scripts 462
 - modifying globally 370
 - modifying locally (severity level) 371
- escape sequence 408
- excluding elements, arrays 32
- Expect property, Web checks 604

F

- failed bitmap synchronization
 - RDP 315, 344
- Federation scenario 191
- fetching data 364
- field demarcation characters 776
- FIELDTBLS environment setting 810
- filter files, editing for .NET 548
- Filter Manager, working with 541
- filtering
 - Java methods 258
 - server traffic scripts 84
- filters in .NET
 - defining 537
 - determining elements to include 536
 - guidelines for setting 535
 - Impact log 541
 - managing 541
 - manipulating 541
 - selecting 539
 - setting 539
- Firefox support 563
- flags, text search 599
- Flash remoting 477
- FLDTBLDIR environment setting 810

Flex

- RTMP 502
- Flex Vuser scripts
 - about 491
 - correlating 494
 - functions 493
 - recording scripts 491
 - understanding 501
 - viewing 498
 - XML tree query 499
- format
 - of data in display buffer 408
- Forms Listener 683
- Frame property, for object checks (Web) 603
- FTP protocol
 - functions 490
 - recording 489
- functions
 - AMF 480
 - ctx (Citrix) 317
 - DNS 380
 - Flex 493
 - FTP 490
 - imap 797
 - Java 431
 - Irc (COM) 459
 - Ird (Database) 362
 - Ireal (Real Player) 821
 - Irs (WinSock) 387
 - mapi 798
 - mms 822
 - pop3 800
 - sapgui (SAP) 723
 - smtp 801
 - te (RTE) 761

G

- General options
 - Citrix display 306
 - Correlation tab 648
- Get Text tool, Citrix Vuser scripts 329
- Global Unique Identifier (GUID) 445
- grids
 - enabling in .NET 516
 - hiding in .NET 516

viewing 366, 515
GUID 445

H

handler routine, Web Services 217
headers
 SOAP 117
hook files 262
hosted by client, server 530
HTML
 maximum parameter length 661
hypergraphic link step, Web Vusers 581
hypertext link step
 defined 581
 modifying 612

I

IC flag 599
ica files 316
identities element 172
identities, Web Service security 177
IDispatch interface 466
ignore namespaces 52
IIOP 239
image checks
 modifying (Web) 613
 Web Vuser scripts 600
Image Step Properties dialog box 614
image synchronization 337
IMAP protocol 795
Impact log, .NET filter 541
importing
 SOAP requests into script 69
importing services 45
include command, .NET filters 545
incoming traffic 83
Informix 357
init method 430
input arguments, Web Services 97
Instantiating COM objects 463
Internet Messaging (IMAP) 795
iterations
 simulating in Web Services 188
IUnknown interface 445

J

Jacada Vuser scripts
 recording 240
Java
 custom filters 258
Java plug-in 241
Java Vuser scripts 243
 recording 238
Java Vusers
 correlating statements 267
 editing Java methods 429
 environment settings 439
 inserting rendezvous points 434
 programming 427
 recording tips 241
Java Vusers (custom)
 creating template 429
 using Java code 422
JavaScript Vusers 426
JMS
 for Web Services 199
 functions 201
 message type 201
 run-time settings 124
 transport method 198
 understanding 198
JNDI properties
 advanced, context factory 287
 locating EJB home 289
 specifying 286

K

keyboard mapping (RTE) 763

L

LDAP protocol
 functions 504
 recording 503
 via WinSock 382
legacy Web service security 144
libc functions, calling 421
Link Step Properties dialog box 612
load balancing, Oracle NCA 686
logical address 186

Index

Irc functions 459
Ird (Database) functions 362
Ireal functions 821
Irs functions 387

M

Mailing Services protocols
 IMAP 797
 MAPI 798
 POP3 800
 recording 796
 SMTP 801
managing
 Web Services in VuGen 39
MAPI
 working with functions 798
mapping keyboard 763
MatchCase property 603
maximum length of HTML parameters 661
Media Player 822
message signatures 150
methods, Java 429
Microsoft .NET Vuser scripts
 correlating 516
 getting started 511
 limitations 510
 managing filters 541
 manipulating filters 541
 overview 510, 533
 recording 509
 troubleshooting 519
 viewing data grids 515
MIME attachment, .NET 222
MMS functions (MS Media Player) 822
modifying Web scripts
 image steps 613
 rendezvous points 627
 submit data steps 619
 submit form steps 615
 think time 628
 transactions 626
 URL steps 609
MS
 Exchange protocol (MAPI) 798
 SQL Server, recording on 357

MTOM 180
MTS components 455
multilingual support *See Volume I - Using VuGen*

N

namedPipe 191
namespaces, ignore 52
navigating through WinSock data 391
NCA Vusers, *see Oracle NCA*
.NET Vusers, *see Microsoft .NET Vuser scripts*
NET Filters 220
netTcp 191
New button 766
New Virtual User dialog box
 RTE 766
non-printable characters 409

O

ODBC recording 357
offset of data in buffer (WinSock) 405
OnFailure property, Web checks 604
Operations tab 43
optional elements
 excluding 30
optional parameters 104
optional windows 716
optional windows (SAPGUI) 722
options
 general *See Volume I - Using VuGen*
Oracle
 recording 2-tier database 357
Oracle Configurator 683
Oracle NCA Vuser scripts
 check connection mode 684
 correlating 686
 creating 671
 recording guidelines 674
 secure applications 683
 servlet testing 683
 using Vuser functions 681
OrbixWeb 239
OTA, Over-The-Air 829
outgoing traffic 83

output arguments, Web Services 100
Output window 435

P

PAP, Push Access Protocol 828
parameterization
 See Also Volume I - Using VuGen
 Tuxedo scripts 807
 Web Services 28
parameters
 optional 104
Plain SOAP scenario 194
policy files 152
POP3 (Post Office) protocol 800
ports, multiple in Web Service 67
PPG, Push Proxy Gateway 828
private key 179
properties
 AssignToParam (Web) 603
 Expect (Web) 604
 Frame (Web) 603
 get and set for Web Services 219
 MatchCase (Web) 603
 OnFailure (Web) 604
 Repeat (Web) 604
 Report (Web) 604
 text checks 603
Protection Level 181
protocols, *See* Vusers
proxy server
 for WSDLs 43
push support
 Wireless and WAP 828
push technology 561

Q

Quality Center
 importing from 49
 test instances 73
 Web service integration 73

R

raise tolerance 347
rdp

 agent for microsoft terminal server
 351
RDP Agent
 tips 353
 troubleshooting 353
rdp agent 351
RDP Vuser scripts
 recording 337
 synchronizing replay 342
read only WinSock buffers 389
realm, Web Service security 183
RealPlayer 819
recording
 Web Services 60
recording at the cursor 708
recording options
 See Also Volume I - Using VuGen
 WinSock 384
recording Vuser scripts
 AJAX 474
 AMF 477
 CORBA sessions 238
 Database 361
 DNS 379
 Flex 491
 FTP 489
 LDAP 503
 Mailing services 795
 Oracle NCA 673
 SAP (Click and Script) 732
 SAPGUI 693
 SAP-Web 735
 Tuxedo 803
 Window Sockets 381
 Wireless 823
recursive elements, in WSDL 107
reduce tolerance 348
references, adding for .NET 543
regression testing, WSDL 52
Reliable Messaging 180
rendezvous points
 Java Vusers 434
 modifying in Web scripts 627
Repeat property, Web Vusers 604
Report property, Web checks 604
reports

Index

- comparison of XML 54
- Response Buffer Capacity 185
- return codes
 - database 369
- RMI
 - recording over IIOP 239
- roots, multiple 37
- row information, Database Vusers 364
- RTE Vuser scripts
 - getting started 759
 - introducing 758
 - mapping PC keyboard 763
 - overview 757
 - reading text from screen 791
 - recording 765
 - steps in creating 759
 - synchronizing 779
 - using te functions 761
- RTMP 502
- rules
 - adding from Correlation tab 651
 - advanced correlation 638
 - creating from correlation results 646
 - defining for correlation 639
 - testing in correlation 642
- run-time settings
 - See Also Volume I - Using VuGen*
- S**
- safearray log (COM) 458
- SAML
 - signing assertion 152
- SAML options 152
- SAP (Click and Script) Vuser Scripts
 - about 731
- SAPGUI Vuser scripts
 - functions 723
 - inserting steps 710
 - recording 693
 - recording at the cursor 708
 - replaying 721
 - run-time settings 722
 - snapshots 710
 - using sapgui functions 723
- SAPGUI/SAP-Web dual protocol 693
- SAP-Web Vuser scripts
 - recording 735
 - run-time settings 740
- Scan for Correlations command
 - Database Vusers 374
- scenario
 - for Web Service WCF coverage 170
- Script Generator, *See* VuGen
- script view
 - Web Services scripts 27
- searching for text on screen (RTE) 792
- security
 - attributes for Web Services Vusers 143
 - for importing WSDLs 43
 - setting for Web Services 145
 - tokens and encryption 147
 - Web Service 144
 - Web Services customizing 162
- security exceptions, .NET 519
- Security Token Service (STS) 175
- SED utility 560
- serialization (Java correlation) 272
- server traffic
 - creating basic script 82
 - getting started with scripts 79
- Service steps
 - modifying in tree view (Web) 629
 - properties dialog box 629
- Service Test Management 73
- services
 - deleting from list 52
 - management 40
- shifted coordinates 350
- Siebel correlation library 743
- Siebel-Web
 - correlating 634, 743
 - recording 742
 - troubleshooting 752
- signatures, Web Service messages 150
- SMTP protocol 801
- snapshots
 - choosing which to display 92
 - Citrix Vusers 307
 - multiple RDP 346
 - SAPGUI Vusers 710
 - Web Services scripts 90

- Winsock buffer 389
 - XML Vusers 664
 - SOA scripts
 - getting started 22
 - SOA tests
 - parameterizing 28
 - running 121
 - viewing and editing 25
 - SOAP headers 117
 - SOAP requests, importing 69
 - SOAP response, saving 120
 - Solaris
 - ASCII translations 385
 - SPN 177
 - SSL
 - certificates for server traffic script 86
 - SSL, testing Web Service 192
 - streaming data protocols
 - mms functions 822
 - RealPlayer functions 821
 - recording 820
 - STS 175
 - SubjectKeyIdentifier 159
 - Submit Data step
 - defined 581
 - dialog box (Web) 620
 - modifying-Web 619
 - Submit Form step
 - defined 581
 - dialog box 616
 - modifying 615
 - SWECCount, correlating 751
 - synchronization failure
 - Citrix 315
 - RDP 344
 - synchronization functions
 - generating for RDP 342
 - synchronizing images 337
 - synchronizing Vuser scripts
 - block-mode (IBM) terminals 781
 - character-mode (VT) terminals 784
 - overview (RTE) 779
 - waiting for terminal to be silent 788
 - waiting for text to appear (RTE) 786
 - waiting for the cursor to appear 784
 - system variables
 - RTE 786
 - Tuxedo 810
- T**
- te (RTE) functions 761
 - TE system variables 786
 - template
 - Java Vuser 429
 - Terminal Emulation 765
 - Terminal Services
 - Citrix Vusers 320
 - Terminal Setup dialog box 768
 - text
 - reading text from screen (RTE) 792
 - searching for text on screen (RTE) 792
 - text checks
 - defined 591
 - defining additional properties 603
 - flags 599
 - text synchronization
 - RDP 342
 - text view (WinSock) 389
 - think time
 - dialog box (Web treeview) 628
 - modifying in Web scripts 628
 - recommended ratio for Siebel 755
 - threshold, WinSock 387
 - thread, main (Java programming) 442
 - thread-safe code 441
 - tips
 - recording RDP 336
 - Token Substitution Testpad dialog box 642
 - token, parameterizing 636
 - tolerance level (RDP) 347, 348
 - toolkit
 - selecting for Web Services 47
 - traffic information, providing 83
 - traffic on server 77, 131, 225
 - transactions
 - modifying in Web scripts 626
 - Translation table settings 385
 - translation, ASCII on UNIX 385
 - transport layer, configuring 196
 - transport, customizing HTTP 183
 - tree view

Index

- SOA tests 25
- Web Services scripts 25
- troubleshooting, .NET 519
- TUXDIR environment setting 810
- Tuxedo Vuser scripts
 - data buffers 809
 - developing 803
 - environment settings 810
 - log file 808
 - running 808
 - system variables 810
 - understanding 806
 - versions 811
 - viewing data files 809
- typing style (RTE Vusers) 774

U

- UDDI
 - information 45
 - search 48
 - specifying server information 48
- undo buffer, emptying (WinSock) 396
- UPN 177
- URL Step Properties dialog box
 - Web 611
- URL steps
 - defined (Web Vusers) 581
 - modifying 609
- user handler, Web Services 217
- Username (Transport Protection)
 - Authentication 174
- Username Authentication (Message Protection) 174

V

- value sets 30
 - optional elements 30
- VB Vusers 423
- VBScript Vusers 425
- verification checks
 - RTE 791
 - sapgui 716
 - Web (Click and Script) 573

- virtual machine settings 123
- Visigenic 239
- Visual Studio
 - viewing scripts 514
- VM run-time settings, Web Services 123
- Vuser functions
 - AMF 480
 - ctx (Citrix) 317
 - DNS 380
 - Flex 493
 - FTP 490
 - imap 797
 - Java 431
 - lrc (COM) 462
 - lrd (Database) 362
 - lreal 821
 - lrs (WinSock) 387
 - mapi 798
 - mms (MS Media Player) 822
 - Oracle NCA 681
 - pop3 800
 - sapgui (SAP) 723
 - smtp 801
 - te (RTE) 761
- Vuser Generator, *See* VuGen
- Vuser information, obtaining (Java) 435
- Vuser scripts
 - C support 421
 - custom 417
 - Java language recording 231
 - programming 417
 - server traffic 77, 131, 225
 - streaming data 819
- Vuser types
 - Java 243
 - COM 462
 - EJB testing 279
 - Java (programming) 427
 - list of 18
 - Media Player 819
 - Real Player 819

W

waiting, for terminal to stabilize(RTE) 788

WCF 163

Web (Click and Script) Vuser scripts

about 554

adding steps 607

deleting steps 608

modifying 605

troubleshooting tips 572

verifying text and images 589

Web correlation 631

Web Service calls

adding 93

adding new 67

properties 93

snapshots 90

viewing snapshot and properties 89

Web Service security

adding 144

customizing 162

Web Services

negative testing 131, 225

Specifications 163

WCF 163

Web Services security, Federation 175

Web Services Vuser scripts

adding content 60

getting started 22

managing services 39

message signatures 150

parameterizing 28

recording 60

running 121

snapshots 90

viewing and editing 25

Web to Java converter 560

Web Vuser scripts

adding steps 607

checks 589

correlating 634

custom request steps 623

deleting steps 608

functions 579

image checks 600

introducing 551

modifying 605

sections 559

understanding 553

verifying text and images 589

wildcards, Citrix window names 318

Windows Authentication 173

Windows Sockets Protocol 381

Windows Sockets Vuser scripts

data buffers 402

data files 403

excluding sockets 386

getting started 383

script and tree view 382

using lrs functions 387

viewing data files 402

Windows Store 178

Winsock Protocol 381

WinSock recording options 384

Wireless Vuser scripts

getting started 825

recording 823

workflow

for Web services 65

WS-Addressing 206

WS-Addressing version 194

WSDL

list of operations 43

refreshing 52

viewing 57

WSDL documents

attachments 113

comparing 52

regression testing 52

WSFederationHttpBinding 181

WS-Reliable Messaging 180

WS-SecureConversation 188

WS-Security 180

WS-Security, customizing 162

WSxxx Tuxedo variables 810

X

X.509 certificate 177

X.509 certificates 182

XML

comparing files 56

custom requests 667

Index

- editing tree in Web Services 118
 - parameterizing elements 28
 - testing 663
- XML arrays 101
- XML editing 29
- XP window style, Citrix 305
- XSD
 - Any type 130
- X-SYSTEM message (RTE) 781