

ProTune™
Creating Virtual User Scripts User's Guide
Version 1.5



MERCURY INTERACTIVE

ProTune Creating Virtual User Scripts User's Guide, Version 1.5

This manual, and the accompanying software and other documentation, is protected by U.S. and international copyright laws, and may be used only in accordance with the accompanying license agreement. Features of the software, and of other products and services of Mercury Interactive Corporation, may be covered by one or more of the following patents: U.S. Patent Nos. 5,701,139; 5,657,438; 5,511,185; 5,870,559; 5,958,008; 5,974,572; 6,138,157; 6,144,962; 6,205,122; 6,237,006; 6,341,310; and 6,360,332. Other patents pending. All rights reserved.

Mercury Interactive, the Mercury Interactive logo, WinRunner, XRunner, FastTrack, LoadRunner, LoadRunner TestCenter, TestDirector, TestSuite, WebTest, Astra, Astra SiteManager, Astra SiteTest, Astra QuickTest, QuickTest, Open Test Architecture, POPs on Demand, TestRunner, Topaz, Topaz ActiveAgent, Topaz Delta, Topaz Observer, Topaz Prism, Twinlook, ActiveTest, ActiveWatch, SiteRunner, Freshwater Software, SiteScope, SiteSeer and Global SiteReliance are registered trademarks of Mercury Interactive Corporation or its wholly-owned subsidiaries, Freshwater Software, Inc., and Mercury Interactive (Israel) Ltd. in the United States and/or other countries.

ActionTracker, ActiveScreen, ActiveTune, ActiveTest SecureCheck, Astra FastTrack, Astra LoadTest, Change Viewer, Conduct, ContentCheck, Dynamic Scan, FastScan, LinkDoctor, ProTune, RapidTest, SiteReliance, TestCenter, Topaz AIMS, Topaz Console, Topaz Diagnostics, Topaz Open DataSource, Topaz Rent-a-POP, Topaz WeatherMap, TurboLoad, Visual Testing, Visual Web Display and WebTrace are trademarks of Mercury Interactive Corporation or its wholly-owned subsidiaries, Freshwater Software, Inc. and Mercury Interactive (Israel) Ltd. in the United States and/or other countries. The absence of a trademark from this list does not constitute a waiver of Mercury Interactive's intellectual property rights concerning that trademark.

All other company, brand and product names are registered trademarks or trademarks of their respective holders. Mercury Interactive Corporation disclaims any responsibility for specifying which marks are owned by which companies or which organizations.

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA
Tel: (408) 822-5200
Toll Free: (800) TEST-911, (866) TOPAZ-4U
Fax: (408) 822-5300

© 2003 Mercury Interactive Corporation, All rights reserved

If you have any comments or suggestions regarding this document, please send them via e-mail to documentation@merc-int.com.

Table of Contents

Welcome to ProTune	xix
Online Resources	xix
ProTune Documentation Set	xx
Using the ProTune Documentation Set	xxi
Typographical Conventions.....	xxiii

PART I: INTRODUCING

Chapter 1: Developing Vuser Scripts	3
Introducing Vusers	3
Looking at Vuser Types	5
Developing Vuser Scripts.....	6

PART II: WORKING WITH VUGEN

Chapter 2: Introducing VuGen	11
About VuGen.....	11
Recording Vuser Scripts with VuGen	12
Setting up your VuGen Environment	13
Running Vuser Scripts with VuGen	14
Understanding VuGen Code	15
Using C Vuser Functions	18
Getting Help on Functions.....	21
Chapter 3: Recording with VuGen	25
About Recording with VuGen	25
Vuser Script Sections	26
Recording Vuser Scripts.....	28
Ending a Recording Session.....	36
Importing Actions	39
Regenerating a Vuser Script.....	40

Chapter 4: Selecting a Script Generation Language	43
About Selecting a Script Generation Language	43
Selecting a Script Language	44
Applying the Basic Options.....	45
Understanding the Correlation Options.....	46
Setting the Recording Options	46
Chapter 5: Configuring the Port Mappings	49
About Configuring the Port Mappings	49
Port Mapping Rules	50
Setting the Auto-Detection Options	51
Traffic Forwarding	52
Setting the Port Mapping Recording Options.....	52
Chapter 6: Enhancing Vuser Scripts	59
About Enhancing Vuser Scripts.....	60
Inserting Transactions into a Vuser Script	61
Inserting Rendezvous Points into a Vuser Script	63
Inserting Comments into a Vuser Script.....	64
Obtaining Vuser Information	65
Sending Messages to Output	66
Handling Errors in Vuser Scripts During Execution	69
Synchronizing Vuser Scripts.....	70
Emulating User Think Time	71
Handling Command Line Arguments	72
Encrypting Text	72
Using C Functions in Vuser Scripts.....	73
Chapter 7: Defining Parameters	75
About Defining Parameters	76
Understanding Parameter Limitations.....	77
Creating Parameters	78
Defining Parameter Properties	82
Understanding Parameter Types	83
Assigning Internal Data	83
Specifying a Parameter Format.....	92
Selecting a File as a Source for Parameter Values.....	94
Importing Data from Existing Databases	99
User-Defined Functions.....	102
Parameterization Options	105

Chapter 8: Correlating Statements	107
About Correlating Statements	107
Using Correlation Functions for C Vusers	109
Using Correlation Functions for Java Vusers	110
Comparing Vuser Scripts using WDiff	111
Modifying Saved Parameters	113
Chapter 9: Configuring Run-Time Settings	115
About Run-Time Settings	115
Configuring Run Logic Run-Time Settings (multi-action).....	117
Pacing Your Actions	121
Configuring Pacing Run-Time Settings (multi-action)	122
Setting Pacing and Run Logic Options (single action)	124
Configuring the Log Run-Time Settings	125
Configuring the Think Time Settings	129
Configuring Additional Attributes Run-Time Settings	131
Configuring Miscellaneous Run-Time Settings.....	132
Setting the VB Run-Time Settings	137
Chapter 10: Running Vuser Scripts in Stand-Alone Mode	139
About Running Vuser Scripts in Stand-Alone Mode.....	139
Running a Vuser Script in VuGen.....	140
Using VuGen's Debugging Features	143
Using VuGen's Debugging Features for Web Vuser Scripts	144
Working with VuGen Windows.....	147
Running a Vuser Script from a Command Prompt.....	147
Running a Vuser Script from a UNIX Command Line	148
Integrating a Vuser Script into a Session Step.....	149
Chapter 11: Managing Scripts Using TestDirector	151
About Managing Scripts Using TestDirector	151
Connecting to and Disconnecting from TestDirector	151
Opening Scripts from a TestDirector Project	154
Saving Scripts to a TestDirector Project	156

PART III: WORKING WITH JAVA LANGUAGE PROTOCOLS

Chapter 12: Recording Java Language Vuser Scripts 161
About Recording Java Language Vuser Scripts..... 161
Getting Started with Recording..... 162
Understanding Java Language Vuser Scripts..... 163
Running a Script as Part of a Package 164
Viewing the Java Methods 165
Manually Inserting Java Methods 166
Configuring Script Generation Settings..... 169

Chapter 13: Setting Java Recording Options 173
About Setting Java Recording Options..... 173
Java Virtual Machine (JVM) Options 174
Classpath Options 176
Recorder Options 177
Serialization Options 181
Correlation Options 183
Debug Options 184
CORBA Options 187

Chapter 14: Correlating Java Scripts..... 189
About Correlating Java Scripts 189
Standard Correlation 190
Advanced Correlation 190
String Correlation..... 192
Using the Serialization Mechanism 194

Chapter 15: Configuring Java Run-Time Settings 201
About Java Run-Time Settings..... 201
Understanding the Java VM Settings 202
Classpath Options 202
Specifying the Virtual Machine Settings..... 204

PART IV: APPLICATION DEPLOYMENT SOLUTION PROTOCOLS

Chapter 16: Developing Citrix Vuser Scripts	207
About Recording Citrix Vuser Scripts.....	207
Getting Started with Citrix Vuser Scripts.....	208
Setting the Recording Options	210
Synchronizing Replay.....	213
Using Citrix Functions	216
Viewing a Citrix Vuser Script	218
Setting the Citrix Display Settings	220
Setting the Citrix Run-Time Settings	221
Understanding ICA Files	223
Disconnecting from the Citrix Server	224
Tips for Working with Citrix Vuser Scripts.....	226

PART V: CLIENT SERVER PROTOCOLS

Chapter 17: Developing Database Vuser Scripts	231
About Recording Database Vuser Scripts	232
Introducing Database Vusers.....	232
Understanding Database Vuser Technology	233
Getting Started with Database Vuser Scripts	234
Setting Database Recording Options.....	235
Using LRD Functions.....	239
Understanding Database Vuser Scripts	244
Evaluating Error Codes.....	249
Handling Errors	250
Chapter 18: Correlating Database Vuser Scripts	253
About Correlating Database Vuser Scripts	253
Scanning a Script for Correlations	254
Correlating a Known Value.....	256
Database Correlation Functions.....	258
Correlating Siebel Scripts.....	259
Chapter 19: Developing DNS Vuser Scripts	263
About Developing DNS Vuser Scripts	263
Working with DNS Functions	264
Chapter 20: Developing WinSock Vuser Scripts	265
About Recording Windows Sockets Vuser Scripts.....	265
Getting Started with Windows Sockets Vuser Scripts	266
Setting the Recording Options	268
Using LRS Functions.....	271
Switching Between Tree View and Script View.....	275

Chapter 21: Working with Window Sockets Data	277
About Working with Windows Socket Data	277
Viewing Data in the Snapshot Window	278
Navigating Through the Data	279
Modifying Buffer Data.....	283
Modifying Buffer Names	288
Viewing Windows Socket Data in Script View.....	289
Understanding the Data File Format.....	291
Viewing Buffer Data in Hexadecimal format	293
Setting the Display Format.....	295
Debugging Tips.....	298
Manually Correlating WinSock Scripts.....	299

PART VI: CUSTOM

Chapter 22: Creating Custom Vuser Scripts	305
About Creating Custom Vuser Scripts.....	305
C Vusers	307
Java Vusers.....	309
VB Vusers.....	310
VBScript Vusers.....	311
JavaScript Vusers	312
Chapter 23: Programming Java Scripts	313
About Programming Java Scripts	313
Creating a Java Vuser	314
Editing a Java Vuser Script	315
VuGen's Java API.....	317
Working with Java Vuser Functions	320
Setting your Java Environment	326
Running Java Vuser Scripts	326
Compiling and Running a Script as Part of a Package.....	327
Programming Tips	328

PART VII: DISTRIBUTED COMPONENT PROTOCOLS

Chapter 24: Recording COM Vuser Scripts	333
About Recording COM Vuser Scripts	333
COM Overview	334
Getting Started with COM Vusers.....	335
Selecting COM Objects to Record	337
Setting COM Recording Options	338

Chapter 25: Understanding COM Vuser Scripts	345
About COM Vuser Scripts.....	345
Understanding VuGen COM Script Structure.....	346
Examining Sample VuGen COM Scripts.....	348
Scanning a Script for Correlations	354
Correlating a Known Value	356
Chapter 26: Understanding COM Vuser Functions	359
About COM Vuser Functions	360
Creating Instances	360
IDispatch Interface Invoke Method	361
Type Assignment Functions	361
Variant Types.....	362
Assignment from Reference to Variant	363
Parameterization Functions	364
Extraction from Variants.....	366
Assignment of Arrays to Variants.....	366
Array Types and Functions.....	366
Byte Array Functions	368
ADO RecordSet Functions	368
Debug Functions	369
VB Collection Support.....	369
Chapter 27: Developing Corba-Java Vuser Scripts	371
About Corba-Java Vuser Scripts.....	371
Recording a Corba-Java Vuser	372
Working with Corba-Java Vuser Scripts.....	375
Recording on Windows XP and Windows 2000 Server.....	377
Chapter 28: Developing RMI-Java Vuser Scripts	379
About Developing RMI-Java Vuser Scripts	379
Recording RMI over IIOP	380
Recording an RMI Vuser.....	380
Working with RMI Vuser Scripts.....	383
 PART VIII: E-BUSINESS PROTOCOLS	
Chapter 29: Developing FTP Vuser Scripts	387
About Developing FTP Vuser Scripts.....	387
Working with FTP Functions	388
Chapter 30: Developing LDAP Vuser Scripts	391
About Developing LDAP Vuser Scripts.....	391
Working with LDAP Functions	392
Defining Distinguished Name Entries.....	395

Chapter 31: Creating Web Vuser Scripts	397
About Developing Web Vuser Scripts	397
Introducing Web Vusers.....	398
Understanding Web Vuser Technology	399
Getting Started with Web Vuser Scripts.....	399
Recording a Web Session.....	401
Converting Web Vuser scripts into Java	402
Chapter 32: Using Web Vuser Functions	403
About Web Vuser Functions	403
Adding and Editing Functions	404
Web Function List	405
Viewing Scripts in the Tree View	412
Viewing Vuser Scripts in Script View	414
Chapter 33: Recording Web/WinSock and SOAP Vuser Scripts	417
About Recording Web/WinSock Vuser Scripts.....	417
Getting Started with Web/WinSock Vuser Scripts	419
Setting Browser and Proxy Recording Options	420
Setting Web Trapping Recording Options	423
Recording a Web/WinSock Session	425
Recording Palm Applications	427
Chapter 34: Setting Recording Options for Internet Protocols	431
About Setting Recording Options for Internet Protocols.....	431
Working with Proxy Settings	432
Setting Advanced Recording Options	435
Setting a Recording Scheme	437
Chapter 35: Setting Recording Options for Web Vusers	443
About Setting Recording Options	443
Specifying which Browser to Use for Recording	444
Selecting a Recording Mode	445
Recording in HTML-Based Mode	445
Recording in URL-Based Mode.....	450
Specifying the Information to Record.....	453
Chapter 36: Configuring Internet Run-Time Settings	457
About Internet Run-Time Settings	457
Setting Proxy Options	459
Setting Browser Emulation Properties.....	462
Performing HTML Compression	465
Setting The Network Speed	466
Setting Preferences	467
Obtaining Debug Information	471

Chapter 37: Checking Web Page Content	473
About Checking Web Page Content	473
Setting the ContentCheck Run-Time Settings	474
Defining a ContentCheck Rule	475
Chapter 38: Verifying Web Pages Under Load	477
About Verification Under Load	477
Adding a Text Check	480
Using Other Text Check Methods.....	483
Adding an Image Check	485
Defining Additional Properties	488
Using Regular Expressions	489
Chapter 39: Modifying Web and Wireless Vuser Scripts	493
About Modifying Web and Wireless Vuser Scripts	493
Adding a Step to a Vuser Script	494
Deleting Steps from a Vuser Script.....	496
Modifying Action Steps	496
Modifying Control Steps	510
Modifying Service Steps	513
Modifying Web Checks (Web only).....	514
Chapter 40: Configuring Correlation Rules for Web Vuser Scripts ..	515
About Correlating Statements	515
Understanding the Correlation Methods.....	517
Choosing a Correlation Handling Method	522
Testing Rules.....	522
Setting the Correlation Recording Options	523
Chapter 41: Correlating Vuser Scripts After Recording	527
About Correlating with Snapshots.....	527
Understanding Snapshots	528
Setting Up VuGen for Correlation	533
Performing a Scan for Correlations.....	536
Performing Manual Correlation.....	539
Defining a Dynamic String's Boundaries	544
Chapter 42: Testing XML Pages	547
About Testing XML Pages.....	547
Viewing XML as URL steps.....	548
Inserting XML as a Custom Request	549
Viewing XML Custom Request Steps	550

Chapter 43: Using Reports to Debug Vuser Scripts	553
About Using Reports to Debug Vuser Scripts	553
Understanding the Results Summary Report	555
Filtering Report Information	556
Searching Your Results	557
Managing Execution Results	558
Chapter 44: Power User Tips for Web Vusers	561
Security Issues.....	561
Handling Cookies.....	564
The Run-Time Viewer (Online Browser)	567
Browsers.....	568
Configuration Issues.....	572

PART IX: ENTERPRISE JAVA BEAN PROTOCOLS

Chapter 45: Performing EJB	575
About EJB Testing.....	575
Working with the EJB Detector.....	576
Creating an EJB Testing Vuser.....	581
Setting EJB Recording Options.....	584
Understanding EJB Vuser Scripts.....	585
Running EJB Vuser Scripts.....	591

PART X: ERP/CRM PROTOCOLS

Chapter 46: Creating Oracle NCA Vuser Scripts	597
About Creating Oracle NCA Vuser Scripts	597
Getting Started with Oracle NCA Vusers	599
Recording Guidelines	600
Enabling the Recording of Objects by Name	601
Using Oracle NCA Vuser Functions	605
Understanding Oracle NCA Vusers	609
Switching Between Tree View and Script View.....	610
Configuring the Run-Time Settings	611
Testing Oracle NCA Applications.....	614
Correlating Oracle NCA Statements for Load Balancing	617
Additional Recommended Correlations	619
Recording in Pragma Mode	621

Chapter 47: Creating Baan Vuser Scripts	623
About Developing Baan Vuser Scripts.....	623
Getting Started with Baan Vuser Scripts	624
Baan Vuser Functions.....	624
Creating a Baan Vuser Script.....	628
Understanding Baan Vuser Scripts.....	629
Customizing Baan Vuser Scripts.....	630

PART XI: LEGACY PROTOCOLS

Chapter 48: Introducing RTE Vuser Scripts	635
About Developing RTE Vuser Scripts	635
Introducing RTE Vusers.....	636
Understanding RTE Vuser Technology	636
Getting Started with RTE Vuser Scripts.....	637
Using TE Functions	638
Mapping Terminal Keys to PC Keyboard Keys.....	640
Chapter 49: Recording RTE Vuser Scripts	643
About Recording RTE Vuser Scripts.....	643
Creating a New RTE Vuser Script	644
Recording the Terminal Setup and Connection Procedure	645
Recording Typical User Actions	648
Recording the Log Off Procedure	649
Setting the Recording Options.....	650
Typing Input into a Terminal Emulator	653
Generating Unique Device Names.....	656
Setting the Field Demarcation Characters	657
Chapter 50: Configuring RTE Run-Time Settings	659
About Terminal Emulator Run-Time Settings	659
Modifying Connection Attempts.....	660
Specifying an Original Device Name	660
Setting the Typing Delay.....	661
Configuring the X-System Synchronization.....	661
Chapter 51: Synchronizing RTE Vuser Scripts	663
About Synchronizing Vuser Scripts.....	663
Synchronizing Block-Mode (IBM) Terminals.....	664
Synchronizing Character-Mode (VT) Terminals	668
Chapter 52: Reading Text from the Terminal Screen	673
About Reading Text from the Terminal Screen	673
Searching for Text on the Screen	673
Reading Text from the Screen	674

PART XII: MAILING SERVICES PROTOCOLS

Chapter 53: Developing Vuser Scripts for Mailing Services.....677
About Developing Vuser Scripts for Mailing Services.....677
Getting Started with Mailing Services Vuser Scripts.....678
Working with IMAP Functions.....679
Working with MAPI Functions.....681
Working with POP3 Functions.....683
Working with SMTP Functions.....684

PART XIII: MIDDLEWARE PROTOCOLS

Chapter 54: Developing Jacada Vuser Scripts689
About Jacada Vuser Scripts.....689
Getting Started with Jacada Vusers.....690
Recording a Jacada Vuser.....691
Replaying a Jacada Vuser.....693
Understanding Jacada Vuser Scripts.....694
Working with Jacada Vuser Scripts.....695

Chapter 55: Developing Tuxedo Vuser Scripts.....697
About Tuxedo Vuser Scripts.....697
Getting Started with Tuxedo Vuser Scripts.....698
Using LRT Functions.....699
Understanding Tuxedo Vuser Scripts.....704
Viewing Tuxedo Buffer Data.....706
Defining Environment Settings for Tuxedo Vusers.....707
Debugging Tuxedo Applications.....708
Correlating Tuxedo Scripts.....708

PART XIV: STREAMING DATA PROTOCOLS

Chapter 56: Developing Streaming Data Vuser Scripts717
About Recording Streaming Data Virtual User Scripts.....717
Getting Started with Streaming Data Vuser Scripts.....718
Using RealPlayer LREAL Functions.....719
Using Media Player MMS Functions.....720

PART XV: WIRELESS PROTOCOLS

Chapter 57: Introducing Wireless Vusers	725
About Wireless Vusers	725
Understanding the WAP Protocol.....	726
Understanding the i-mode System.....	728
i-mode versus WAP.....	729
Understanding VoiceXML.....	729
Chapter 58: Recording Wireless Vuser Scripts	733
About Recording Wireless Vuser Scripts	733
Getting Started with Wireless Vuser Scripts.....	734
Using Wireless Vuser Functions	735
Troubleshooting Wireless Vuser Scripts.....	738
Chapter 59: Working with WAP Vuser Scripts	739
About WAP Vusers.....	739
Recording Over a Phone.....	740
Bearers Support.....	741
RADIUS Support	742
Push Support	742
ProTune Push Support.....	744
MMS Support.....	745
Chapter 60: Setting Wireless Vuser Recording Options	747
About Setting Recording Options	747
Specifying the Recording Mode (WAP only)	748
Specifying the Information to Record (i-mode and VoiceXML)	749
Specifying a Toolkit.....	750
Chapter 61: Configuring WAP Run-Time Settings	751
About WAP Run-Time Settings	751
Configuring Gateway Options	752
Configuring Bearer Information	755
Configuring RADIUS Connection Data	758

PART XVI: INFORMATION FOR ADVANCED USERS

Chapter 62: Creating Vuser Scripts in Visual Studio	763
About Creating Vuser Scripts in Visual Studio.....	763
Creating a Vuser Script with Visual C.....	764
Creating a Vuser Script with Visual Basic	766
Configuring Runtime Settings and Parameters.....	767

Chapter 63: Programming with the XML API	769
About Programming with the XML API	769
Understanding XML Documents	770
Using XML Functions.....	772
Specifying XML Function Parameters.....	775
Working with XML Attributes	777
Structuring an XML Script	777
Enhancing a Recorded Session	779
Chapter 64: VuGen Debugging Tips	783
General Debugging Tip	783
Using C Functions for Tracing	783
Adding Additional C Language Keywords	784
Examining Replay Output.....	785
Debugging Database Applications	785
Working with Oracle Applications	787
Solving Common Problems with Oracle Applications	787
Two-tier Database Scripting Tips.....	791
Chapter 65: Advanced Topics	801
Files Generated During Recording	801
Files Generated During Replay	803
Specifying the Vuser Behavior	805
Command Line Parameters.....	806
Examining the .dat Files.....	806
Adding a New Vuser Type	807

PART XVII: APPENDIXES

Appendix A: The Java Environment: A Comprehensive Guide	815
About the Java Environment	815
Terminology	816
JDK Versions	818
Browsers.....	823
Java Plug-In.....	826
Other Environments	828
Frequently Asked Questions.....	829
Appendix B: EJB Architecture and Testing	833
What are EJBs?.....	833
EJB Architecture.....	833
EJB Structure and Mechanism	834
Deployment in Common Application Servers.....	837
EJB Unit-Testing	838

Appendix C: Calling External Functions	839
Loading a DLL—Locally	839
Loading a DLL—Globally	841
Appendix D: Programming Scripts on UNIX Platforms	843
About Programming Vuser Scripts to Run on UNIX Platforms	843
Generating Templates	844
Programming Vuser Actions into a Script.....	845
Configuring Vuser Run-Time Settings	847
Defining Transactions and Rendezvous Points.....	852
Compiling Scripts	852
Appendix E: Using Keyboard Shortcuts	855
Index	857

Welcome to ProTune

Welcome to ProTune, Mercury Interactive's tool for Tuning in Production testing.

The main intention of ProTune is to enable the user to explore his network, detect bottlenecks and assist during the tuning phase. ProTune combines the network topology, predefined tuning sessions and goals to a set of tests that pinpoint problematic components in the client network. In addition, ProTune helps the more advanced user to explore and tune the business processes by providing a simple and organized methodology.

Online Resources



ProTune includes the following online tools:

Read Me First provides last-minute news and information about ProTune.

Books Online displays the complete documentation set in PDF format. Online books can be read and printed using Adobe Acrobat Reader, which is included in the installation package. Check Mercury Interactive's Customer Support Web site for updates to ProTune online books.

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

ProTune Context Sensitive Help provides immediate answers to questions that arise as you work with ProTune. It describes dialog boxes, and shows you how to perform ProTune tasks. To activate this help, click in a window

and press F1. Check Mercury Interactive's Customer Support Web site for updates to ProTune help files.

Technical Support Online uses your default web browser to open Mercury Interactive's Customer Support Web site. This site enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is <http://support.mercuryinteractive.com>.

Support Information presents the locations of Mercury Interactive's Customer Support Web site and home page, and a list of Mercury Interactive's offices around the world.

Mercury Interactive on the Web uses your default Web browser to open Mercury Interactive's home page (<http://www.mercuryinteractive.com>). This site enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more.

ProTune Documentation Set

ProTune is supplied with a set of documentation that describes how to:

- install ProTune
- create scripts
- use the ProTune Console
- use the ProTune Analysis

Using the ProTune Documentation Set

The ProTune documentation set consists of one installation guide, a Console user's guide, an Analysis user's guide, and two guides for creating Virtual User scripts.

Installation Guide

For instructions on installing ProTune, refer to the *ProTune Installation Guide*. The installation guide explains how to install:

- ▶ the ProTune Console—on a Windows-based machine
- ▶ Virtual User components—for both Windows and UNIX platforms

Console User's Guide

The ProTune documentation pack includes one Console user's guide:

The *ProTune Console User's Guide* describes how to create and run ProTune sessions using the ProTune Console in a Windows environment. The Vusers can run on UNIX and Windows-based platforms. The Console user's guide presents an overview of the ProTune testing process.

Analysis User's Guide

The ProTune documentation pack includes one Analysis user's guide:

The *ProTune Analysis User's Guide* describes how to use the ProTune Analysis graphs and reports after running a session in order to analyze system performance.

Guide for Creating Scripts

The ProTune documentation pack includes one ProTune *ProTune Creating Virtual User Scripts User's Guide*. This guide describes how to create scripts using VuGen. When necessary, supplement this document with the *Online Function Reference*. (**Help > Function Reference**)

For information on	Look here...
Installing ProTune	<i>ProTune Installation Guide</i>
The ProTune testing process	<i>ProTune Console User's Guide</i>
Creating scripts	<i>ProTune Creating Virtual User Scripts User's Guide</i>
Creating and running sessions	<i>ProTune Console User's Guide</i>
Analyzing test results	<i>ProTune Analysis User's Guide</i>

Typographical Conventions

This book uses the following typographical conventions:

1, 2, 3	Bold numbers indicate steps in a procedure.
►	Bullets indicate options and features.
>	The greater than sign separates menu levels (for example, File > Open).
Stone Sans	The Stone Sans font indicates names of interface elements on which you perform actions (for example, “Click the Run button.”).
Bold	Bold text indicates method or function names
<i>Italics</i>	<i>Italic</i> text indicates method or function arguments, file names or paths, and book titles.
Arial	The Helvetica font is used for examples and text that is to be typed literally.
<>	Angle brackets enclose a part of a file path or URL address that can vary (for example, < <i>Product installation folder</i> >/bin).
[]	Square brackets enclose optional arguments.
{ }	Curly brackets indicate that one of the enclosed values must be assigned to the current argument.
...	In a line of syntax, an ellipsis indicates that more items of the same format may be included.

Part I

Introducing

1

Developing Vuser Scripts

Welcome to ProTune, the proactive solution for optimizing production systems. ProTune's system-wide approach to optimization is a product of Mercury Interactive's expert knowledge and tuning methodologies.

ProTune provides a recommended menu of steps that will comprise the foundation of your tuning session. To expand the functionality of ProTune's recommended scripts, you can also use VuGen to create custom scripts for your application.

ProTune emulates an environment in which users concurrently work with your system. To do this, ProTune replaces the human user with a *virtual user* (Vuser). The actions that a Vuser performs are described in a Vuser script. ProTune supplies a variety of tools to help you develop your Vuser scripts.

This chapter includes:

- Introducing Vusers
- Looking at Vuser Types
- Developing Vuser Scripts
- Using this Guide

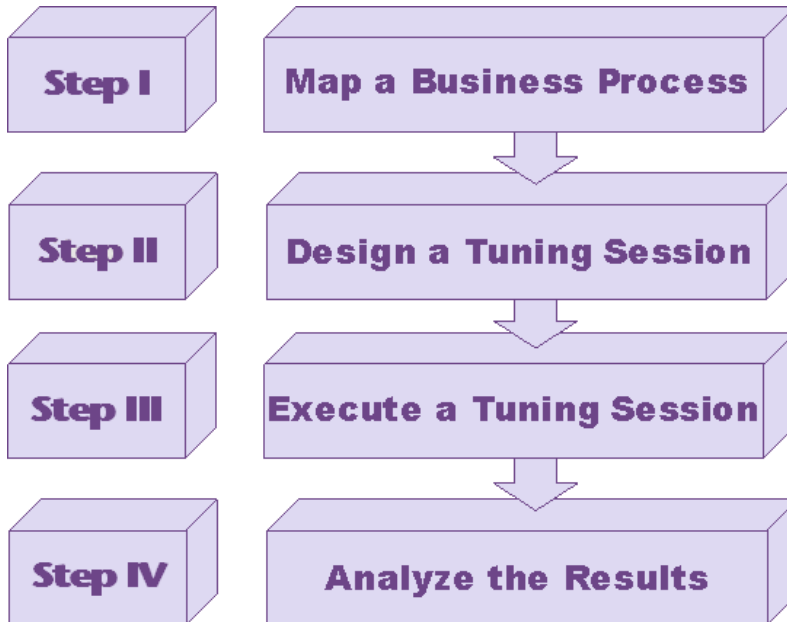
Introducing Vusers

ProTune replaces human users with *virtual users* or *Vusers*. Vusers emulate the actions of human users by performing typical business processes. For each action that a Vuser performs, ProTune submits input to a server or to a similar enterprise system. By changing the number of Vusers, you vary the load on the system. While a workstation accommodates only a single human user, many Vusers can run concurrently on a single workstation.

Using ProTune, you divide your system performance testing requirements into session steps. A session step defines the events that occur during each tuning session. Thus, for example, a session step defines and controls the number of users to emulate, the actions that they perform, and the machines on which they run their emulations.

ProTune has a variety of Vuser types, each type suited to a particular tuning topology. This enables you to use Vusers to accurately model and emulate real world situations. The actions that a Vuser performs during the session step are described in a *Vuser script*. The Vuser scripts include functions that measure and record the performance of the server during the session step. Each Vuser type requires a particular type of Vuser script. Creating the Vuser scripts required for a session step is part of the ProTune testing process.

The chart below shows the four steps of the ProTune tuning process. This guide describes the process of creating custom scripts for the design step of the tuning session. For details about the other steps, refer to the *ProTune Console User's Guide*.



Looking at Vuser Types

VuGen provides a variety of Vuser technologies that enable you to tune various types of servers. Each Vuser technology is suited to a particular architecture and results in a specific type of Vuser. For example, you use Web Vusers to emulate users operating Web browsers; Tuxedo Vusers to emulate Tuxedo clients communicating with a Tuxedo application server; RTE Vusers to operate terminal emulators. The various Vuser technologies can be used alone or together, to create effective tuning session steps.

The Vuser types are divided into the following categories:

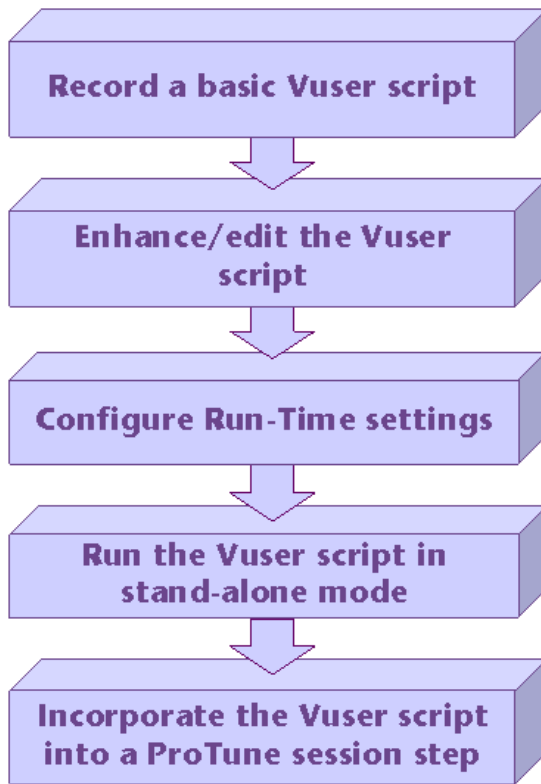
- ▶ **Application Deployment Solution:** Citrix ICA.
- ▶ **Client/Server:** For DB2 CLI, DNS, Informix, MS SQL Server, ODBC, Oracle (2-tier), Sybase Ctlib, Sybase Dblib, and Windows Sockets protocols.
- ▶ **Custom:** For C templates, Visual Basic templates, Java templates, Javascript and VBScript type scripts.
- ▶ **Distributed Components:** For COM/DCOM, Corba-Java and Rmi-Java protocols.
- ▶ **E-business:** For FTP, LDAP, Palm, SOAP, Web (HTTP/HTML), and Web/WinSocket Dual Protocol.
- ▶ **Enterprise Java Beans:** For EJB Testing and Rmi-Java protocols.
- ▶ **ERP:** For Baan, Oracle NCA, Peoplesoft-Tuxedo, SAPGUI, Siebel-DB2 CLI, Siebel MSSQL, and Siebel Oracle protocols.
- ▶ **Legacy:** For Terminal Emulation (RTE).
- ▶ **Mailing Services:** Internet Messaging (IMAP), MS Exchange (MAPI), POP3, and SMTP.
- ▶ **Middleware:** Jacada and the Tuxedo (6, 7) protocols.
- ▶ **Streaming Data:** Media Player (MMS) and Real protocols.
- ▶ **Wireless:** For i-Mode, VoiceXML, and WAP protocols.

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

Developing Vuser Scripts

The structure and content of a Vuser script differ from one Vuser type to another. For example, Database Vuser scripts always have three sections, are written in a code that resembles C, and include SQL calls to a database server. In contrast, GUI Vuser scripts have only one section, and are written in TSL (test script language).

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



You begin the process of developing a Vuser script by recording a basic script. ProTune provides you with a number of tools for recording Vuser scripts (see the table below for a list of the tools). You enhance the basic script by adding control-flow structures and other ProTune API into the

script. You then configure the run-time settings. The run-time settings include iteration, log, and timing information, and define how the Vuser will behave when it executes the Vuser script. To verify that the script runs correctly, you run it in stand-alone mode. When your script runs correctly, you incorporate it into a ProTune session step.

Using this Guide

This guide is divided into several parts:

- ▶ Part I, “Introducing,” is applicable to all types of Vuser scripts.
- ▶ Part II, “Working with VuGen,” is applicable only to those Vuser scripts that are recorded and/or run using VuGen.
- ▶ Parts III to XV apply to specific Vuser script types. Refer to the Table of Contents to locate the part describing your Vusertype.
- ▶ Part XVI contains information for advanced users. It provides some general debugging tips, describes the files generated by VuGen, and explains how to program scripts in Visual C and Visual Basic.
- ▶ Part XVII contains several appendixes with technology overviews. It describes the Java environment, EJB architecture, Calling External Functions, Programming in UNIX, and Keyboard Shortcuts.

Part II

Working with VuGen

2

Introducing VuGen

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

This chapter describes:

- ▶ Recording Vuser Scripts with VuGen
- ▶ Setting up your VuGen Environment
- ▶ Running Vuser Scripts with VuGen
- ▶ Understanding VuGen Code
- ▶ Using C Vuser Functions
- ▶ Getting Help on Functions

The following information applies to all types of Vuser scripts.

About VuGen

The Vuser Script Generator, also known as *VuGen*, is ProTune's primary tool for developing Vuser scripts.

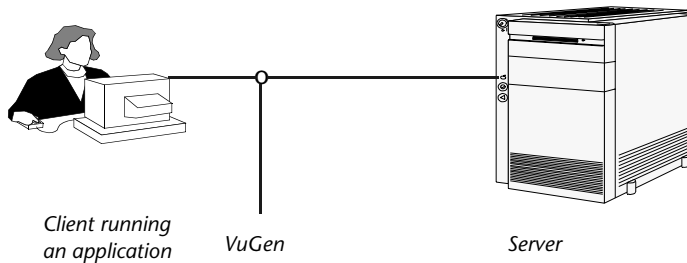
VuGen not only records Vuser scripts, but also runs them. Running scripts from VuGen is useful when debugging. It enables you to emulate how a Vuser script will run when executed as part of a load testing session step.

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

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

Recording Vuser Scripts with VuGen

You use VuGen to develop a Vuser script by recording a user performing typical business processes on a client application. VuGen creates the script by recording the activity between the client and the server. For example, in database applications, VuGen monitors the client end of the database and traces all the requests sent to, and received from, the database server.

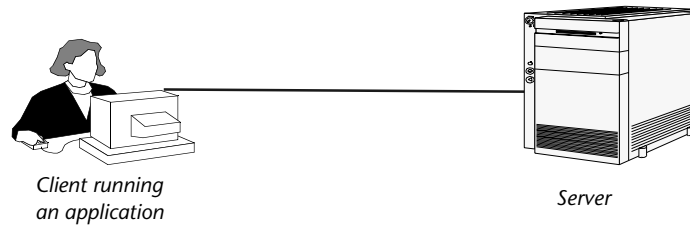


Instead of manually creating a Vuser script by programming the application's API function calls to the server, you use VuGen to:

- ▶ monitor the communication between the application and the server
- ▶ generate the required function calls
- ▶ insert the generated function calls into a Vuser script

Each Vuser script that you create with VuGen can communicate directly with a server by executing calls to the server API—without relying on client

software. You can therefore use Vusers to check server performance even before the user interface of the client software has been fully developed.



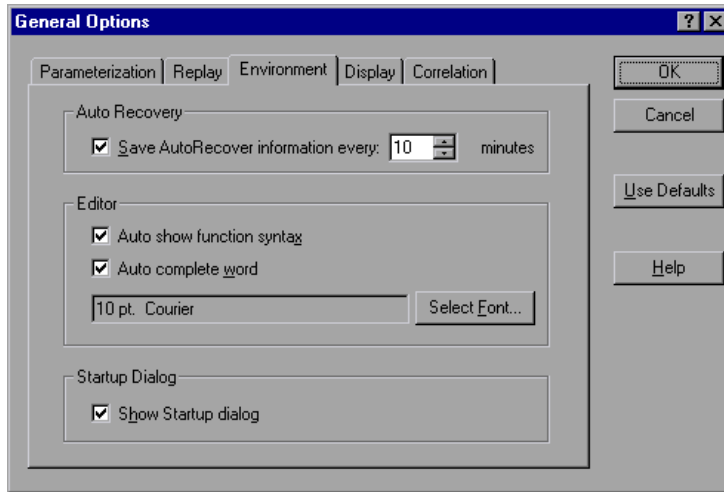
In addition, when a Vuser communicates directly with a server, system resources are not used on a user interface. This lets you run a large number of Vusers simultaneously on a single workstation. This in turn allows you to use only a few testing machines to emulate large server loads.

Setting up your VuGen Environment

You can set up your VuGen working environment in order to customize the auto recovery settings, the VuGen editor, and the startup preferences. Using the auto recovery option, you can restore your Vuser script, settings in the event of a crash or power outage. You can set the editor options for VuGen's Intellisense features which automatically fill in words and function syntax. For more information on these features, see "Getting Help on Functions," on page 21.

To set the environment-related options:

- 1 Select **Tools > General Options** and click the Environment tab.



- 2 To save the current Vuser script, information for auto recovery, select the **Save AutoRecover Information** option and specify the time in minutes between the saves.
- 3 To set the editor font, click **Select Font**. The Font dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys etc.) are available.
- 4 To instruct VuGen to display the Startup Dialog box whenever it opens, select **Show Startup dialog** in the Startup Dialog section.
- 5 Click **OK** to accept the settings and close the General Options dialog box.

Running Vuser Scripts with VuGen

In order to perform load testing with your Vuser script, you need to incorporate the script into a ProTune session step. Before doing this, you can check the script's functionality by running it from VuGen. If the script execution is successful, you can then incorporate it into the session step. For more information on ProTune session steps, refer to your *ProTune Console User's Guide*.

Before you run a Vuser script, you can modify its run-time settings. These settings include the number of iterations that the Vuser performs, and the pacing and the think time that will be applied to the Vuser when the script is run. For more information on configuring run-time settings, see Chapter 9, “Configuring Run-Time Settings.”

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

Advanced users can compile a recorded script to create an executable program. For more information, see Chapter 6, “Enhancing Vuser Scripts.”

Understanding VuGen Code

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

- General Vuser Functions
- Protocol-Specific Vuser Functions

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

General Vuser Functions

The general Vuser functions are also called LR functions because each LR function has an **lr** prefix. The LR functions can be used in any type of Vuser script. The LR functions enable you to:

- Get run-time information about a Vuser, its Vuser Group, and its host.

- Add transactions and synchronization points to a Vuser script. For example, the **lr_start_transaction** (**lr.start_transaction** in Java) function marks the beginning of a transaction, and the **lr_end_transaction** (**lr.end_transaction** in Java) function marks the end of a transaction. See Chapter 6, “Enhancing Vuser Scripts” for more information.
- Send messages to the output, indicating an error or a warning.

See “Using C Vuser Functions,” on page 18 for a list of LR functions, and for details refer to the *Online Function Reference* (**Help > Function Reference**).

Protocol-Specific Vuser Functions

In addition to the general Vuser functions, VuGen also generates and inserts protocol-specific functions into the Vuser script while you record.

The protocol-specific functions are particular to the type of Vuser that you are recording. For example, VuGen inserts LRD functions into a database script, LRT functions into a TUXEDO script and LRS functions into a Windows Sockets script.

By default, VuGen’s automatic script generator creates Vuser scripts in C for most protocols, and Java for Corba-Java/Rmi-Java Vusers. You can instruct VuGen to generate code in Visual Basic or Javascript. For more information, see Chapter 4, “Selecting a Script Generation Language.”

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

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

```
#include "as_web.h"

Action1()
{

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

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

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

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

    lr_start_transaction("Purchase_Order");

    ...
}
```

For more information about using C functions in your Vuser scripts, refer to Chapter 6, “Enhancing Vuser Scripts.” For more information about modifying a Java script, see Chapter 23, “Programming Java Scripts.”

Note: The C Interpreter used for running Vuser scripts only supports the ANSI C language. It does not support any Microsoft extensions to ANSI C.

Using C Vuser Functions

You can add C Vuser functions to any Vuser script in order to enhance the script. VuGen generates only a few of the general Vuser functions while you record. If required, the remaining functions can be manually programmed into a script. For details on the general Vuser functions, see Chapter 6, “Enhancing Vuser Scripts.”

The following list shows the general ProTune functions for ANSI C scripts. This includes all protocols except for Java and GUI. For a list of the Java functions, see Chapter 23, “Programming Java Scripts.”

Transaction Functions:

lr_end_sub_transaction	Marks the end of a sub-transaction for performance analysis.
lr_end_transaction	Marks the end of a ProTune transaction.
lr_end_transaction_instance	Marks the end of a transaction instance for performance analysis.
lr_fail_trans_with_error	Sets the status of open transactions to LR_FAIL and sends an error message.
lr_get_trans_instance_duration	Gets the duration of a transaction instance specified by its handle.
lr_get_trans_instance_wasted_time	Gets the wasted time of a transaction instance by its handle.
lr_get_transaction_duration	Gets the duration of a transaction by its name.
lr_get_transaction_think_time	Gets the think time of a transaction by its name.
lr_get_transaction_wasted_time	Gets the wasted time of a transaction by its name.
lr_resume_transaction	Resumes collecting transaction data for performance analysis.

<code>lr_resume_transaction_instance</code>	Resumes collecting transaction instance data for performance analysis.
<code>lr_set_transaction_instance_status</code>	Sets the status of a transaction instance.
<code>lr_set_transaction_status</code>	Sets the status of open transactions.
<code>lr_set_transaction_status_by_name</code>	Sets the status of a transaction.
<code>lr_start_sub_transaction</code>	Marks the beginning of a sub-transaction.
<code>lr_start_transaction</code>	Marks the beginning of a transaction.
<code>lr_start_transaction_instance</code>	Starts a nested transaction specified by its parent's handle.
<code>lr_stop_transaction</code>	Stops the collection of transaction data.
<code>lr_stop_transaction_instance</code>	Stops collecting data for a transaction specified by its handle.
<code>lr_wasted_time</code>	Removes wasted time from all open transactions.

Command Line Parsing Functions

<code>lr_get_attrib_double</code>	Retrieves a <i>double</i> type variable used on the script command line.
<code>lr_get_attrib_long</code>	Retrieves a <i>long</i> type variable used on the script command line.
<code>lr_get_attrib_string</code>	Retrieves a string used on the script command line.

Informational Functions

<code>lr_user_data_point</code>	Records a user-defined data sample.
<code>lr_whoami</code>	Returns information about a Vuser script to the Vuser script.

<code>lr_get_host_name</code>	Returns the name of the host executing the Vuser script.
<code>lr_get_master_host_name</code>	Returns the name of the machine running the ProTune Console

String Functions

<code>lr_eval_string</code>	Replaces a parameter with its current value.
<code>lr_save_string</code>	Saves a null-terminated string to a parameter.
<code>lr_save_var</code>	Saves a variable length string to a parameter.
<code>lr_save_datetime</code>	Saves the current date and time to a parameter.
<code>lr_advance_param</code>	Advances to the next available parameter.
<code>lr_decrypt</code>	Decrypts an encoded string.
<code>lr_eval_string_ext</code>	Retrieves a pointer to a buffer containing parameter data.
<code>lr_eval_string_ext_free</code>	Frees the pointer allocated by lr_eval_string_ext .
<code>lr_save_searched_string</code>	Searches for an occurrence of string in a buffer and saves a portion of the buffer, relative to the string occurrence, to a parameter.

Message Functions

<code>lr_debug_message</code>	Sends a debug message to the Output window.
<code>lr_error_message</code>	Sends an error message to the Output window.
<code>lr_get_debug_message</code>	Retrieves the current message class.
<code>lr_log_message</code>	Sends a message to a log file.
<code>lr_output_message</code>	Sends a message to the Output window.
<code>lr_set_debug_message</code>	Sets a debug message class.

<code>lr_vuser_status_message</code>	Generates and prints formatted output to the Console Vuser status area.
<code>lr_message</code>	Sends a message to the Vuser log and Output window.
Run-Time Functions	
<code>lr_load_dll</code>	Loads an external DLL.
<code>lr_peek_events</code>	Indicates where a Vuser script can be paused.
<code>lr_think_time</code>	Pauses script execution to emulate think time—the time a real user pauses to think between actions.
<code>lr_continue_on_error</code>	Specifies an error handling method.
<code>lr_rendezvous</code>	Sets a rendezvous point in a Vuser script.

Getting Help on Functions

You can get help for ProTune functions in several ways:

- ProTune Function Reference
- IntelliSense
- Header File

ProTune Function Reference

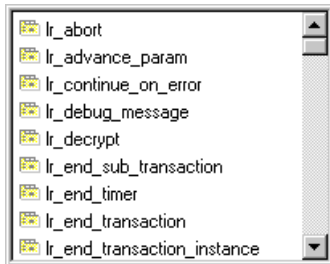
The online *Online Function Reference* contains detailed syntax information about all of the ProTune functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

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

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

IntelliSense

The VuGen editor now incorporates *Intellisense*, also known as *Word Completion*. When you begin typing a function, the Intellisense feature opens a list box displaying all available matches to the function prefix.



To use a function, select it. VuGen places it at the location of the cursor. To close the list box, press Esc.

To instruct VuGen to use this feature globally, choose **Tools > General Options** and select the Environment tab. Select the check box adjacent to the **Auto complete word** option. The list box opens when you type the first underscore. By default, word completion is enabled globally.

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

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

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

To disable this feature, choose **Tools > General Options** and select the Environment tab. Clear the check box adjacent to the **Auto show function syntax** option. If you disable **Show Function Syntax** globally, you can still bring up the syntax by pressing Ctrl+Shift+Space or choosing **Edit > Show Function Syntax** after typing the opening parenthesis in the editor.

Header File

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

In most cases, the name of the header file corresponds to the prefix of the protocol. For example, Database functions that begin with an **lrd** prefix, are listed in the *lrd.h* file. The following table shows the header files that are associated with the most commonly used protocols:

Protocol	File
Citrix	<i>ctrxfuncs.h</i>
COM/DCOM	<i>lrc.h</i>
Database	<i>lrd.h</i>
FTP	<i>mic_ftp.h</i>
General C function	<i>lrun.h</i>
IMAP	<i>mic_imap.h</i>
LDAP	<i>mic_mldap.h</i>
MAPI	<i>mic_mapi.h</i>
MediaPlayer	<i>mic_media.h</i>
Oracle NCA	<i>orafuncs.h</i>
POP3	<i>mic_pop3.h</i>
RealPlayer	<i>lreal.h</i>
SAPGUI	<i>as_sap.gui.h</i>

Protocol	File
Siebel	<i>lrdsiebel.h</i>
SMTP	<i>mic_smtp.h</i>
Terminal Emulator	<i>lrrte.h</i>
Tuxedo	<i>lrt.h</i>
Web	<i>as_web.h</i>
Windows Sockets	<i>lrs.h</i>

3

Recording with VuGen

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

This chapter describes:

- ▶ Vuser Script Sections
- ▶ Recording Vuser Scripts
- ▶ Ending a Recording Session
- ▶ Importing Actions
- ▶ Regenerating a Vuser Script

The following information applies to all types of Vuser scripts.

About Recording with VuGen

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

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

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

While recording, you can insert transactions, comments, and rendezvous points into the script. For details, see Chapter 6, “Enhancing Vuser Scripts.”

Vuser Script Sections

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

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

When you run multiple iterations of a Vuser script, only the *Actions* sections of the script are repeated—the *vuser_init* and *vuser_end* sections are not repeated. For more information on the iteration settings, see Chapter 9, “Configuring Run-Time Settings.”

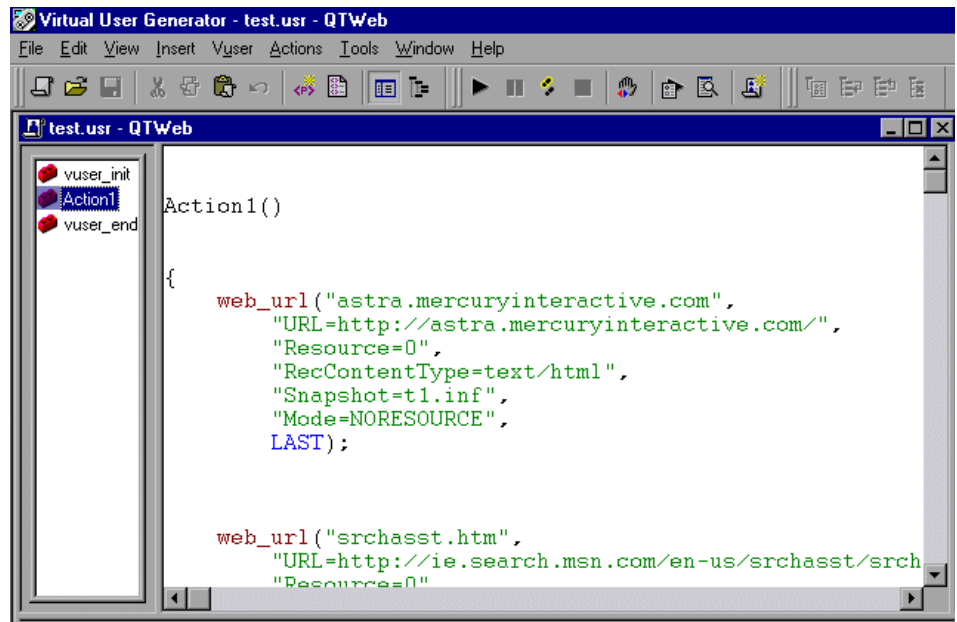
You use the VuGen script editor to display and edit the contents of each of the script sections. You can display the contents of only a single section at a time. To display a section, highlight its name in the left pane.

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

the *end* method. For more information, see Chapter 23, “Programming Java Scripts.”

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

In the following example, the VuGen script editor displays the *Action1* section of a Web Vuser script.



Recording Vuser Scripts

VuGen allows you to record in a single or multi-protocol mode. When you record a single protocol, VuGen only records the specified protocol. When you record in multi-protocol mode, VuGen records the actions in several protocols. Multi-protocol scripts are supported for the following protocols: COM, FTP, IMAP, Oracle NCA, POP3, RealPlayer, Window Sockets (raw), SMTP, and Web. The engine for the *Dual protocol Web/Ws* uses a different mechanism and should be treated as a single protocol—it may not be combined with the other multi-protocol types.

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

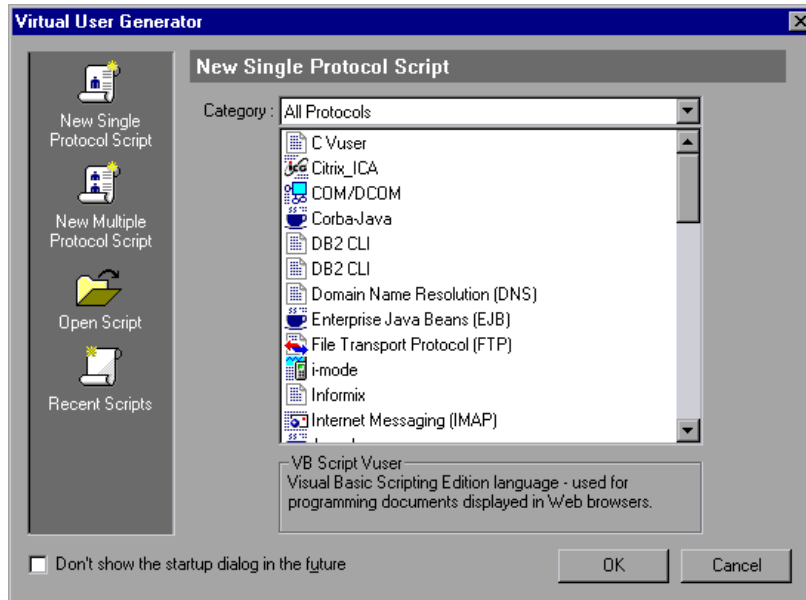
For most Vuser types, you create a new Vuser script each time you record—you cannot record into an existing script. However, when recording a Java, CORBA-Java, RMI-Java, Web, WAP, i-mode, Voice XML, Oracle NCA, or RTE Vuser script, you can also record within an existing script.

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

For all Java language Vusers (CORBA, RMI, Jacada, and EJB) refer to Chapter 12, “Recording Java Language Vuser Scripts” for details about recording, or the chapter discussing the specific protocol.

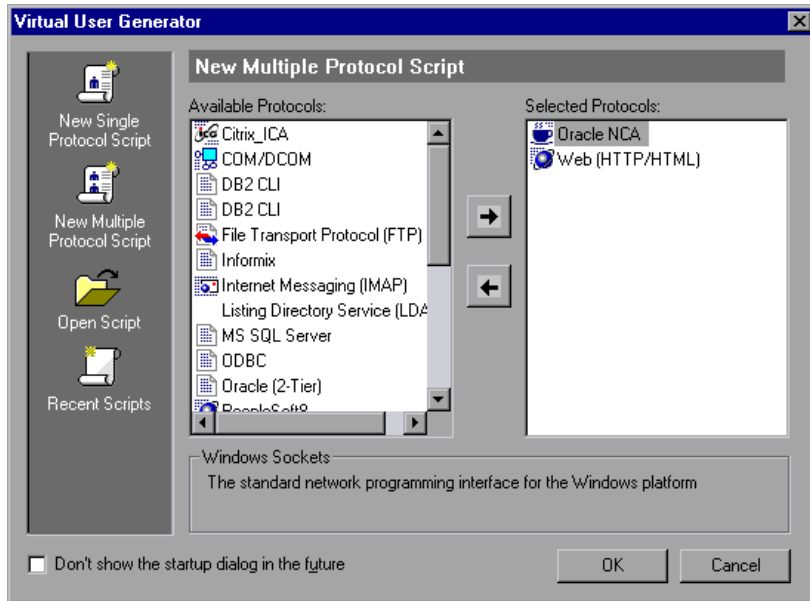
To create a new Vuser script:

- 1 Start VuGen. The startup screen opens (unless you disabled it when you last opened VuGen).



- 2 To create a single protocol script, make a selection from the Category list and select one of the protocols.

- 3 To create a multi-protocol script, allowing you to record two or more protocols in a single recording session, click the **New Multiple Protocol Script** button in the left pane to enable the Protocol Selection window.



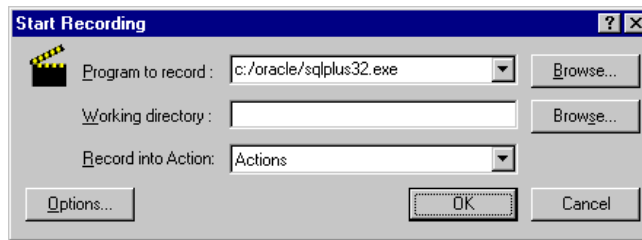
Select the desired protocol from the **Available Protocols** list. Click the right-facing arrow to move the selection into the **Selected Protocols** list. Repeat this step for all of the desired protocols.

Note: When recording certain Oracle NCA applications, you only need to choose **Oracle NCA**—not **Web Protocol**. For details, see Chapter 46, “Creating Oracle NCA Vuser Scripts.” For SOAP, use the Web/WinSock dual protocol from the Single Protocol list.

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

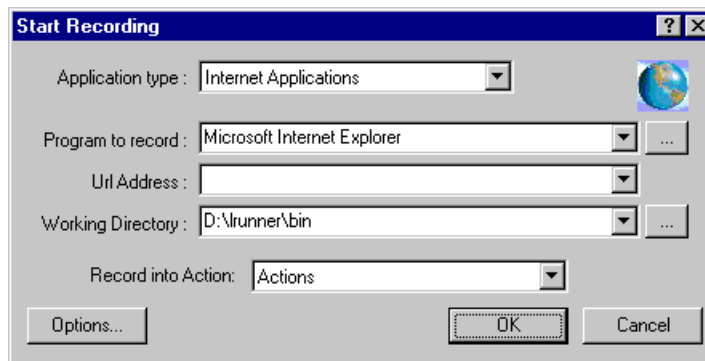


- 6 For most Vuser script types, VuGen automatically opens the Start Recording dialog box when you create the new script. If the Start Recording dialog box was not opened, click the **Start Recording** button. The Start Recording dialog box opens. This dialog box will differ, based on the protocol you are recording.
- 7 For most Client/Server protocols, the following dialog box opens:



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

- 8 For non-Internet applications, choose the application type: Win32 Applications or Internet Applications. For example, Web and Oracle NCA scripts record Internet Applications, while Windows Socket Vusers records a Win32 application.
- 9 For Internet Applications, fill in the relevant information:

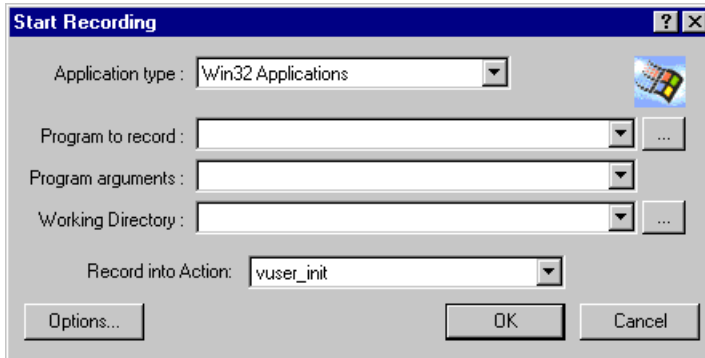


Program to record: Select the browser or Internet application to record. For Citrix, locate *rundlg.exe* file in the bin directory.

URL Address: Specify the starting URL address.

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

- 10 For Win32 Applications, fill in the relevant information:

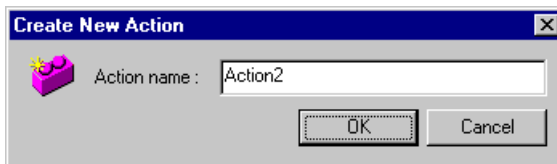


Program to record: Enter the Win 32 application to record.

Program Arguments: Specify command line arguments for the executable specified above. For example, if you specify *plus32.exe* with the command line options *peter@neptune*, it connects the user *Peter* to the server *Neptune* when starting *plus32.exe*.

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

- 11 In the **Record into Action** box, select the section into which you want to record. Initially, the available sections are *vuser_init*, *Action1*, and *vuser_end*. For single-protocol Vuser scripts that support multiple actions (Oracle NCA, Web, RTE, C Vusers, WAP, i-Mode, and VoiceXML), you can add a new section by selecting **Actions > Create New Action** and specify a new action name.



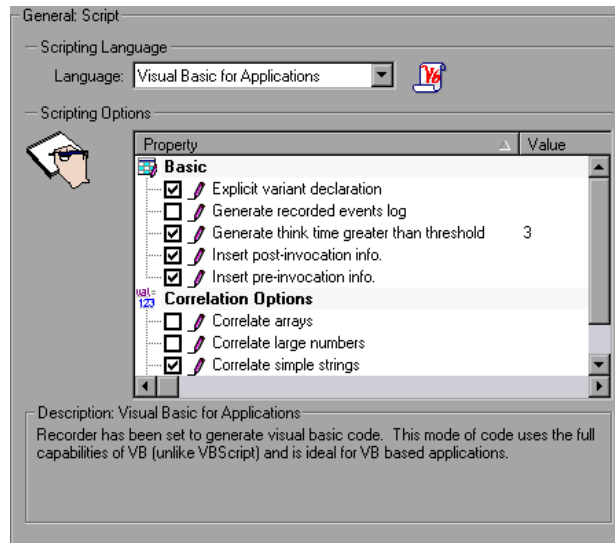
12 To record the application startup, select **Record the application startup** (not applicable to Java type Vuser script). To instruct VuGen not to record the application startup, clear the check box. In the following instances, it may not be advisable to record the startup:

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

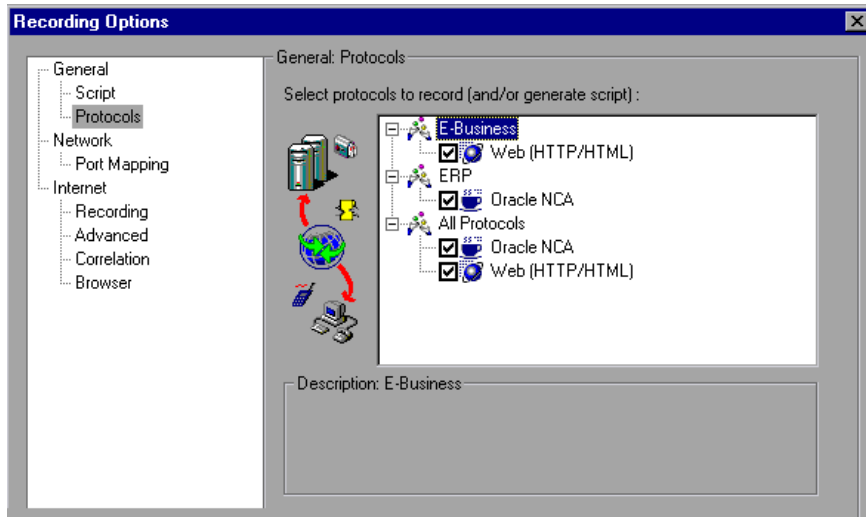


13 Click **Options** or the **Recording Options** button to set the recording options. You can set recording options in the following areas: Port Mapping, Browser (Oracle NCA only), Protocols (multi-protocol only), and Script.

14 Click the **Script** tab to choose a language for code generation and to set the scripting options. For details, see Chapter 4, “Selecting a Script Generation Language.”

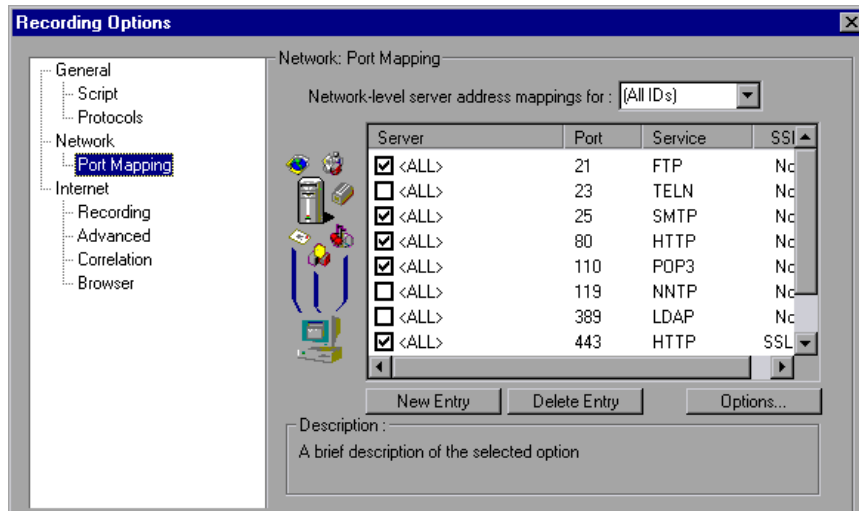


- 15** For a multi-protocol Vuser script only: To modify the protocols that you want to record, click the **Protocol** tab. Expand the node and select the check boxes adjacent to the desired protocols.

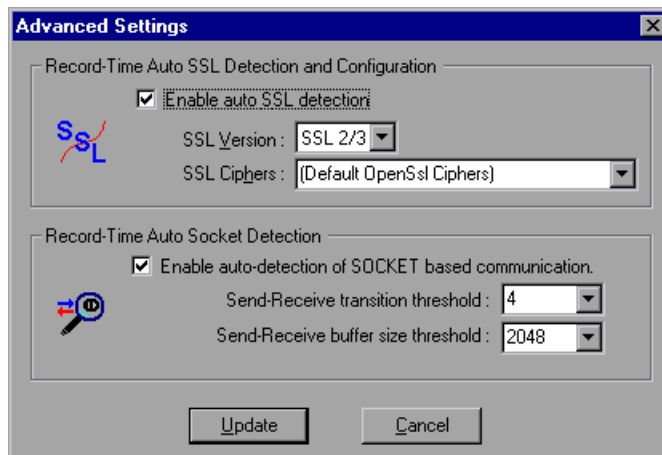


- 16** To specify port information, click the **Port Mapping** tab. This is useful when recording SSL applications on a non-standard port. Review the list of ports. If the port you are using is not on the list, click **New Entry**. Enter the port number in the **Target Port** box and choose the applicable type in the

Connection Type box (for example SSL). For more information, see the *Creating Vuser Scripts* guide.



- 17** To enable SSL detection for a secure recording, click **Options**. The Advanced Settings dialog box opens.



Select the **Enable auto SSL detection** check box and specify an SSL version and SSL cipher.



- 18 Click **Options** or the **Recording Options** button to set the applicable recording options. For general recording options, see Chapter 4, “Selecting a Script Generation Language” and Chapter 5, “Configuring the Port Mappings.” For other protocol-specific recording options, see their respective chapters.
- 19 Click **OK** to close the dialog box and begin recording.
- 20 If you cleared the **Record the application startup** check box, the Recording Suspended dialog box appears. When you reach the point at which you want to start recording, click **Record**. If you decide not to record, click **Abort**.
- 21 VuGen starts your application and the Recording toolbar appears.



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

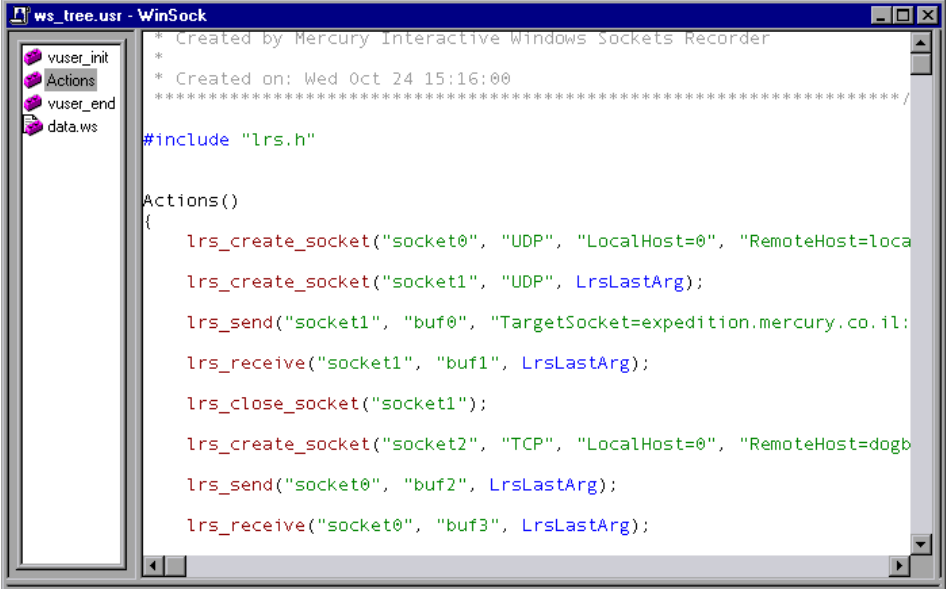
Ending a Recording Session

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

To complete the recording:

- 1 Switch to the *vuser_end* section in the floating toolbar, and perform the log off or cleanup procedure.

- Click the **Stop Recording** button on the Recording toolbar. The VuGen editor displays all the recorded statements.



```

ws_tree.usr - WinSock
* Created by Mercury Interactive Windows Sockets Recorder
* Created on: Wed Oct 24 15:16:00
*****

#include "lrs.h"

Actions()
{
    lrs_create_socket("socket0", "UDP", "LocalHost=0", "RemoteHost=loca
    lrs_create_socket("socket1", "UDP", LrsLastArg);
    lrs_send("socket1", "buf0", "TargetSocket=expedition.mercury.co.il:
    lrs_receive("socket1", "buf1", LrsLastArg);
    lrs_close_socket("socket1");
    lrs_create_socket("socket2", "TCP", "LocalHost=0", "RemoteHost=dogb
    lrs_send("socket0", "buf2", LrsLastArg);
    lrs_receive("socket0", "buf3", LrsLastArg);
  
```

- Click **Save** to save the recorded session. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name. **Note:** Do not name the script *init*, *run* or *end*, since these names are used by VuGen.
- To save the entire script directory as a zip file, choose **File > Export to Zip File**.

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

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

Click **OK**.

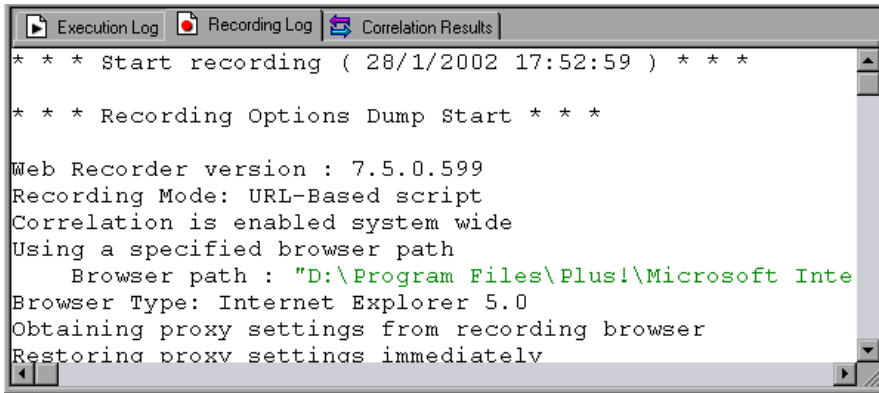
- To create a zip file and send it as an email attachment, choose **File > Zip and Email**.

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

Enter an email address and send your email.

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

To view a log of the messages that were issued during recording, choose **View > Output Window** and select the **Recording Log** tab. You can set the level of detail for this log in the **Advanced** tab of the Recording options.



While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script.

You can manually edit a script in the VuGen editor. For protocols that support multiple actions, you can record additional actions at any time.

Importing Actions

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

To import actions into the current script:

- 1 Select **Actions > Import Action into Vuser**. The Import Action dialog box opens.



- 2 Click **Browse** to select a Vuser script. A list of the script's actions appears in the **Actions to Import** section.
- 3 Highlight an action and click **OK**. The action appears in your script.
- 4 To rearrange the order of actions, you must first enable action reordering. Right-click on any action and select **Enable Action Reorder**. Then drag the actions to the desired order.

Regenerating a Vuser Script

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

In addition, you can parameterize the script and correlate variables. For more information, see Chapter 7, “Defining Parameters.”

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

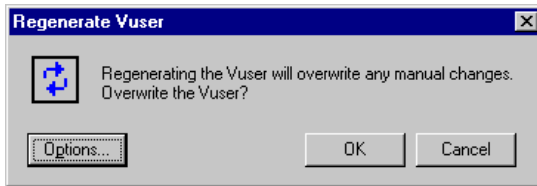
When you regenerate a script, it removes all of the manually added enhancements to the recorded actions. Note that if you added parameters to your script, VuGen restores the original values. The parameter list, however, is not deleted; you can reinsert parameters that you created earlier.

To regenerate a Vuser script, choose **Tools > Regenerate Vuser**. VuGen issues a warning indicating that all manual changes will be overwritten. Note that regeneration, only cleans up the recorded actions—not those that were manually added.

When working with multi-protocol scripts, you can control which protocols to regenerate.

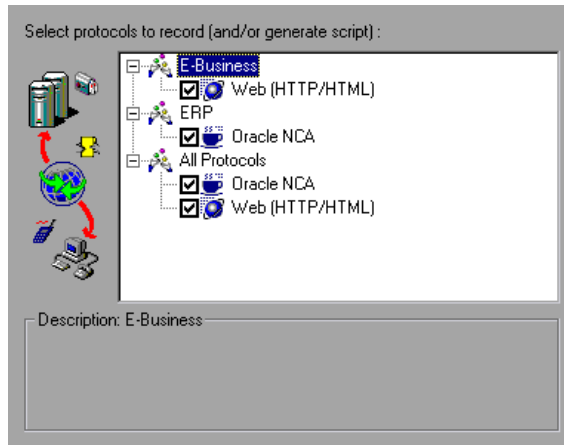
To regenerate a multi-protocol Vuser script:

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

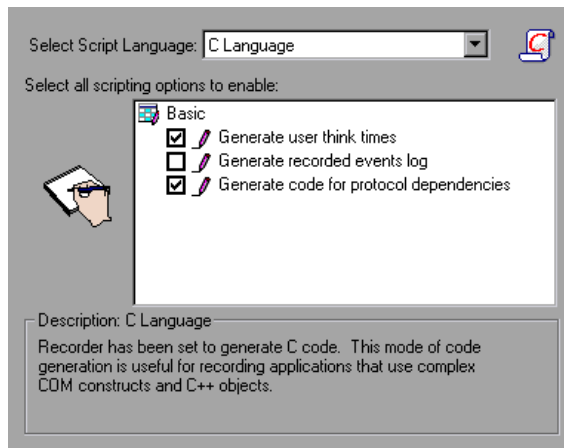


- 2 Click **Options** to open the Regenerate Options dialog box.
- 3 Select the **General:Protocols** node. Indicate which protocols to regenerate and which to leave as is. Select the check boxes of the protocols you want to

regenerate. Clear the check boxes of the protocols you want to leave unchanged.



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



4

Selecting a Script Generation Language

Before you record a script with VuGen, you indicate the desired scripting language: C, Visual Basic, VB Script, or Javascript. This chapter describes the script language recording options that apply to many of the supported protocols.

- Selecting a Script Language
- Applying the Basic Options
- Understanding the Correlation Options
- Setting the Recording Options

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

About Selecting a Script Generation Language

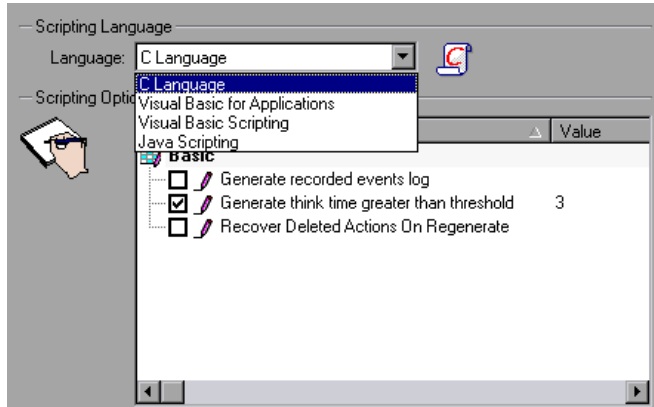
Before you record a session, you can set several recording options which instruct the recorder what to include in the script and how to generate it.

If at least one of the protocols you are recording has multi-protocol capabilities, the *Script* options will be available. The only exception is when you record HTTP or WinSock as a single protocol script. In this case, the *Script* options are not available.

Selecting a Script Language

When you record a session, by default VuGen creates a script that emulates your actions. The default script generation language is C. For the FTP, COM/DCOM, and mail protocols (IMAP, POP3, and SMTP), VuGen can also generate a script in Visual Basic, VB Script, and Javascript.

Before recording, you select a recording language:



C Language - For recording applications that use complex COM constructs and C++ objects.

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

Visual Basic Scripting - For VBscript-based applications, such as ASP.

Java Scripting - For Javascript-based applications such as *js* files and dynamic HTML applications.

After the recording session, you can modify the script with regular C, Visual Basic, VB Script, or Javascript code or control flow statements.

The following sections describe the scripting options. For C scripts you can only set the Basic options. For non-C scripts, you can set additional Basic and Correlation options.

Applying the Basic Options

The Basic script options apply to all generation languages. These options allow you to control the level of detail in the generated script.

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

Generate user think times greater than threshold - Use a threshold value for think time. If the recorded think time is less than the threshold, VuGen does not generate a think time statement. You also specify the threshold value. The default value is 3—if the think time is less than 3 seconds, VuGen does not generate think time statements. If you clear this check box, VuGen will not generate any think times.

Split action section to functions by event count - Create a new action function if the number of recorded events exceeds the specified threshold.

Recover Deleted Actions on Regenerate - When regenerating scripts, (**Tools > Regenerate Vuser**), recover deleted actions, provided that there are no existing actions with the same name. (disabled by default)

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

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

Explicit variant declaration - Declare variant types explicitly in order to handle *ByRef* variants. (disabled by default)

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

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

Understanding the Correlation Options

Correlation allows you to save dynamic values during test execution. These settings let you configure the extent of automatic correlation performed by VuGen while recording. All of correlation options are disabled by default. The Correlation options only apply to the VBScript and JScript languages.

Correlate small numbers - Correlate short data types such as bytes, characters, and short integers. (disabled by default)

Correlate large numbers - Correlate long data types such as integers, long integers, 64-bit characters, float, and double. (disabled by default)

Correlate simple strings - Correlate simple, non-array strings and phrases. (enabled by default)

Correlate arrays - Track and correlate arrays of all data types, such as string, structures, numbers, etc. (disabled by default)

Correlate structures - Track and correlate complex structures. (disabled by default)

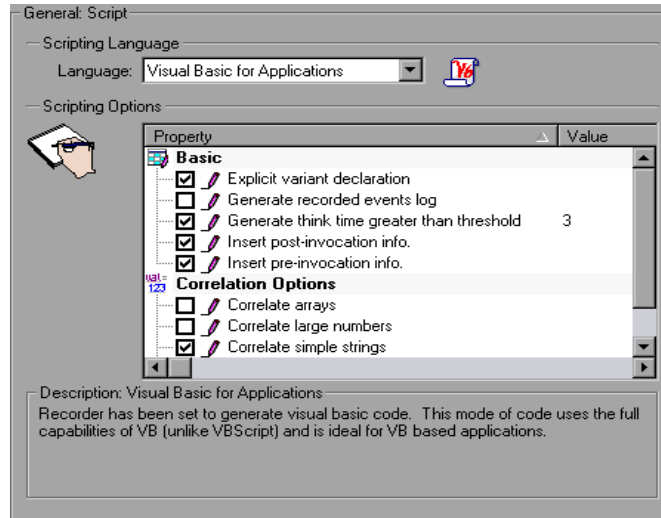
Setting the Recording Options

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

To set the script recording options:

- 1 Open the Recording Options. Choose **Tools > Recording Options** from the main menu or click **Options...** in the Start Recording dialog box. The Recording Options dialog box opens.

2 Select the **General:Script** node.



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

5

Configuring the Port Mappings

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

This chapter describes:

- ▶ Port Mapping Rules
- ▶ Setting the Auto-Detection Options
- ▶ Traffic Forwarding
- ▶ Setting the Port Mapping Recording Options

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

About Configuring the Port Mappings

When recording Vuser scripts that record network traffic on a socket level (HTTP, SMTP, POP3, FTP, IMAP, Oracle NCA and WinSocket), you can set the Port Mapping options. Using these options, you can map the traffic from a specific server:port combination to the desired communication protocol.

The available communication protocols to which you can map are FTP, HTTP, IMAP, NCA, POP3, SMTP, and SOCKET. You create a mapping by specifying a server name, port number, or a complete server:port combination. For example, you can indicate that all traffic from the server *twilight* on port 25, should be handled as SMTP. You can also specify that all traffic from the server called *viper*, should be mapped to the FTP protocol, regardless of the port. Additionally, you can map all traffic on port 23 to SMTP, regardless of the server name.

When recording in multi-protocol mode, if at least one of the protocols records on a socket level, the *Port Mapping* options will be available. The only exception is when you record HTTP or WinSock as a single protocol script. In this case, the *Port Mapping* options are not available.

Port Mapping Rules

VuGen uses the following priorities in assigning data to a service:

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

A map entry with a high priority does not get overridden by an entry with a lower priority. For example, if you specify that traffic on server *twilight* using port 25 be handled as SMTP and then you specify that all servers on port 25 be handled as HTTP, the data will be treated as SMTP.

In addition, the following guidelines apply:

- **Port 0:** Port number 0 indicates any port.
- **Forced mapping:** If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol to record that communication, even though the actual communication may be HTTP. In this instance, the Vuser script might be empty.

After you define a port mapping, it appears in the list of Port Mappings. You can temporarily disable any entry by clearing the check box adjacent to it. When you disable an entry, VuGen ignores all traffic to that server:port combination. You should disable the port entry when the data is irrelevant or if the protocol is not supported.

Setting the Auto-Detection Options

By default, no mappings are defined and VuGen employs *auto-detection*. VuGen's auto-detection analyzes the data that is sent to the server. It checks the data for a signature, a pattern in the data's content, that identifies the protocol. For the purpose of detecting a signature, all of the send buffers until the first receive buffer, are combined. All send buffers that were sent until a receive buffer is returned, are considered a single data *transition*. In some protocols, VuGen determines the type in a single transition, (such as HTTP). Other network protocols require several transitions before determining the type. For this purpose, VuGen creates a temporary buffer, per server-port combination. If VuGen cannot determine the protocol type by reading the first transition buffers, it stores the data in a temporary buffer. It continues to read the incoming buffers until it detects a signature of a specific protocol.

By default, VuGen allows 4 transitions and uses a temporary buffer of 2048 bytes in order to detect a protocol signature. If VuGen has not yet determined the type after reaching the maximum number of transitions, or after reaching the maximum buffer size, it assigns the data to the WinSock protocol. If, you did not instruct VuGen to record the WinSock protocol (in the multi-protocol selection), VuGen discards the data.

In the Advanced Settings dialog box, you can change the maximum number of buffers you want VuGen to read in order to detect the protocol type. You can also specify the size of the temporary buffer. In instances where the amount of data in the first send buffers, is greater than the size of the temporary buffer, VuGen cannot auto-detect the protocol type. In this case, you should increase the size of the temporary buffer.

When working with the above network level protocols, it is recommended that you allow VuGen to use auto-detection to determine the protocol type. In most cases, VuGen's recorder is able to recognize the signatures of these protocols. It then automatically processes them according to the protocol specifications. In certain instances, however, VuGen may be unable to recognize the protocol. For example:

- The protocol signature closely resembles an existing protocol, resulting in erroneous processing.
- There is no unique signature for the protocol.

- ▶ The protocol uses SSL encryption, and therefore cannot be recognized on a WinSock level.

In all of the above cases, you can supply information to uniquely identify the server and port hosting the protocol.

Traffic Forwarding

VuGen allows you to forward all traffic from a specific port to another server. This is particularly useful in cases where VuGen cannot run properly on the client, such as unique UNIX machines, or instances where it is impossible to launch the application server through VuGen. We configure VuGen to intercept the traffic from the problematic client machine, and pass it on to the server. In this way, VuGen can process the data and generate code for the actions.

For example, if you were working on a UNIX client called *host1*, which communicated with a server, *server1*, over port 8080, you would create a Port Mapping entry for *server1*, port 8080. In the **Traffic Forwarding** section of the Server Entry dialog box, you enable traffic forwarding by selecting the **Allow forwarding to target server from local port** check box. You specify the port from which you want to forward the traffic, in our example 8080.

You then connect the client, *host1*, to the machine running VuGen, instead of *server1*. VuGen receives the communication from the client machine and forwards it via the local port 8080, to the server. Since the traffic passes through VuGen, it can analyze it and generate the appropriate code.

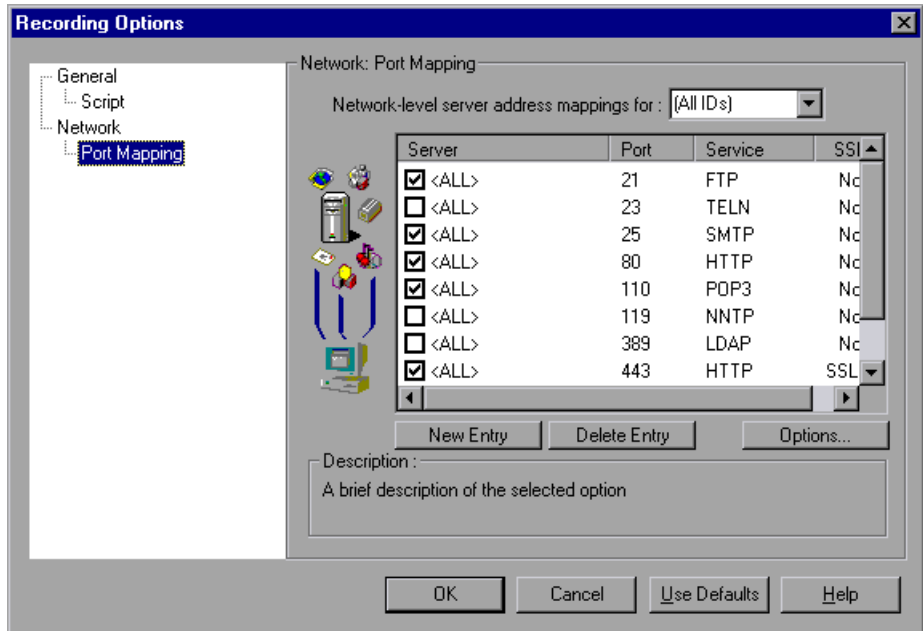
Setting the Port Mapping Recording Options

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

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

To set the port mapping recording options:

- 1 Choose **Tools > Recording Options** from the main menu or click **Options...** in the Start Recording dialog box. The Recording Options dialog box opens. Select the **Network:Port Mapping** node.



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

Server Entry

Socket Service

Service ID : HTTP Service Type : TCP

Target Server : twilight

Target Port : [Any] Connection Type : Auto

SSL Configuration

SSL Version : SSL 2/3

SSL Ciphers : [Default OpenSSL Ciphers]

Use specified client-side certificate (Base64/PEM)

Client Cert : [] ...

Password : []

Use specified proxy-server certificate (Base64/PEM)

Proxy Cert : [] ...

Password : []

Test SSL

Traffic Forwarding

Allow forwarding to target server from local port : []

Description

Server certificate password. A password must be specified if required for the specified server certificate.

Update Cancel

- 3 Enter the following Server information:

Service ID: A protocol or service name used by the recorder to identify the type of connection. (i.e. HTTP, FTP, etc.) You can also specify a new name. The name may not exceed 8 characters.

Service Type: The type of service, currently set to TCP.

Target Server: The IP address or hostname of the target server for which this entry applies. The default is All Servers.

Target Port: The port of the target server for which this entry applies. Port 0 implies all ports.

Connection Type: The security level of the connection: Plain (non-secure), SSL, or Auto. If you select Auto, the recorder checks the first 4 bytes for an SSL signature. If it detects the SSL signature, it assumes that SSL is being used.

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

SSL Version: The preferred SSL version to use when communicating with the client application and the server. By default is SSL 2/3 is used. However some services require SSL 3.0 only or SSL 2.0 only. Some new wireless applications require TLS 1.0—a different security algorithm.

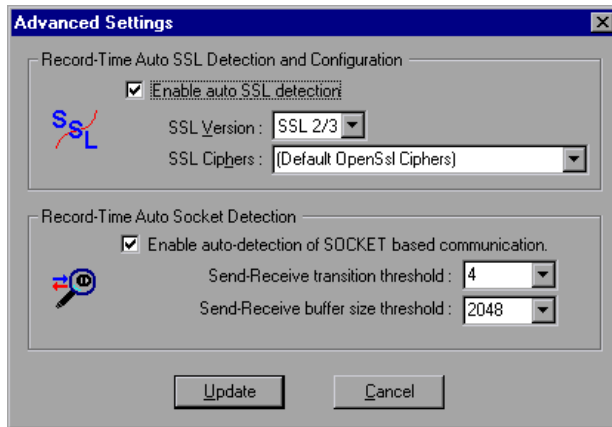
SSL Cipher: The preferred SSL cipher to use when connecting with a remote secure server.

Use specified client-side certificate: The default client-side certificate to use when connecting to a remote server. Specify or browse for a certificate file in *txt*, *crt*, or *pem* format, and supply a password.

Use specified proxy-server certificate: The default server certificate to present to client applications that request a server certificate. Specify or browse for a certificate file in *txt*, *crt*, or *pem* format, and supply a password.

- 5 When you specify a specific server and port, you can test SSL communication to that server:port. Click **Test SSL** to check the authentication information against the server.
- 6 To allow traffic forwarding, select **Allow forwarding to target server from local port**, and specify a port number. Note that this option is only enabled when the **Target Server** and **Target Port** are unique (not <Any>).

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



To automatically detect SSL communication, select the **Enable auto SSL detection** box. Specify the version and default cipher that you want to detect. Note that this only applies to port mappings that were defined as *auto* in the **Connection type** box, or not defined at all. If a server, port, or server:port combination was defined as either Plain or SSL, then auto SSL detection does not apply.

To automatically detect the type of communication, select **Enable auto detection of SOCKET based communication**. If required, raise the maximum number of transitions, one at a time until VuGen succeeds in detecting the protocol. You can also gradually increase the maximum buffer size by 1024 bytes (1 KB) at a time until VuGen succeeds in detecting the protocol. This allows VuGen to review a larger amount of data in order to find a signature.

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

- 8 To view all of the entries, Select **All** in the **Network-level server address mappings** box. The Server Entry dialog box opens. Make the necessary changes in the configuration properties. Click **OK** to close the Server Entry dialog box. Note that you cannot change the server name or port number of an entry. You can only change the connection type and security settings.
- 9 To modify an existing entry, select it and click **Edit Entry**.

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

6

Enhancing Vuser Scripts

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

This chapter describes:

- ▶ Inserting Rendezvous Points into a Vuser Script
- ▶ Inserting Comments into a Vuser Script
- ▶ Obtaining Vuser Information
- ▶ Sending Messages to Output
- ▶ Handling Errors in Vuser Scripts During Execution
- ▶ Synchronizing Vuser Scripts
- ▶ Emulating User Think Time
- ▶ Handling Command Line Arguments
- ▶ Using C Functions in Vuser Scripts

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

About Enhancing Vuser Scripts

While you are recording a Vuser script, or after you record it, you can enhance its capabilities by adding the following types of functions:

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

General Vuser Functions

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

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

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

Protocol-Specific Vuser Functions

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

Standard ANSI C Functions

You can enhance your Vuser scripts by adding standard ANSI C functions. ANSI C functions allow you to add comments, control flow statements, conditional statements, and so forth to your Vuser scripts. You can add standard ANSI C functions to any type of Vuser script. For details, see “Using C Functions in Vuser Scripts,” on page 73.

Inserting Transactions into a Vuser Script

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

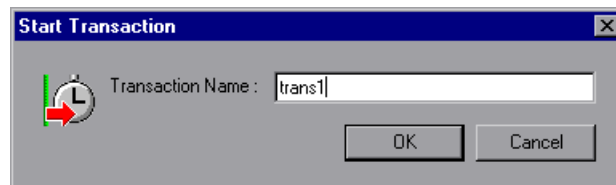
To measure a transaction, you insert Vuser functions to mark the beginning and the end of a task. Within a script, you can mark an unlimited number of transactions, each transaction with a different name.

During session step execution, the Console measures the time that it takes to perform each transaction. After the session step run, you analyze the server’s performance per transaction using ProTune’s graphs and reports.

To mark the start of a transaction:



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



- 2 Type a transaction name in the Transaction Name box. Transaction names must begin with a letter or number and may contain letters, numbers, or the following characters !, \$, %, &, ', -, [, ^, _, ` , <, >, {, }, |, or ~.

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

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

Note: You can insert transactions into your script after you complete a recording session by selecting **Insert > Start Transaction** to mark the beginning of the transaction, and **Insert > End Transaction** to mark its end.

To mark the end of a transaction:



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



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

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

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

Inserting Rendezvous Points into a Vuser Script

To emulate heavy user load on your system, you synchronize Vusers to perform a task at exactly the same moment. You ensure that multiple Vusers act simultaneously by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it is held by the Console until all Vusers participating in the rendezvous arrive. When the rendezvous conditions are met, the Vusers are released by the Console.

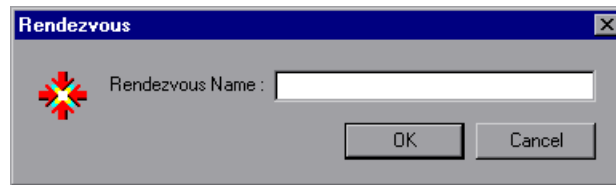
You designate the meeting place by inserting a rendezvous point into your Vuser script. When a Vuser executes a script and encounters the rendezvous point, script execution is paused and the Vuser waits for permission from the Console to continue. After the Vuser is released from the rendezvous, it performs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

Note: You cannot add a rendezvous to the *init* or *end* actions of your script.

To insert a rendezvous point:



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



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

Click **OK** to accept the rendezvous name. VuGen inserts an **lr_rendezvous** statement into the Vuser script. For example, the following function defines a rendezvous point named *rendezvous1*:

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

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

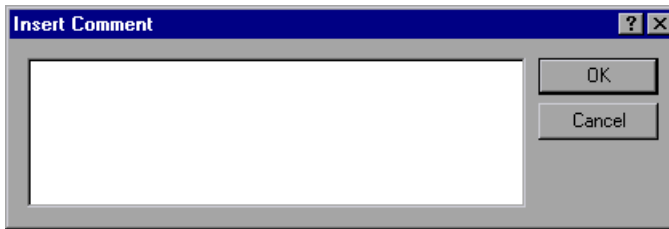
Inserting Comments into a Vuser Script

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

To insert a comment:



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



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

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

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

Obtaining Vuser Information

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

lr_get_attrib_string	Returns a string containing command line argument values or run-time information such as the Vuser ID or the load generator name.
lr_get_host_name	Returns the name of the load generator for the Vuser.
lr_get_master_host_name	Returns the name of the ProTune Console load generator.
lr_whoami	Returns the Vuser ID, Vuser Group for a Vuser

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

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

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

Sending Messages to Output

When you run a session step, the Console's Output window displays information about script execution. You can include statements in a Vuser script to send error and notification messages to the Console. The Console 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 as this may lengthen the transaction execution time and skew the transaction results.

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

lr_debug_message	Sends a debug message to the output window.
lr_error_message	Sends an error message to the output window.
lr_get_debug_message	Retrieves the current message class.
lr_log_message	Sends an output message directly to a file, <i>output.txt</i> , located in the Vuser script directory. This function is useful in preventing output messages from interfering with TCP/IP traffic.
lr_output_message	Sends a message to the Output window.
lr_set_debug_message	Sets a message class for output messages.
lr_vuser_status_message	Generates and prints formatted output to the Console Vuser status area.
lr_message	Sends a message to the Vuser log and Output window.

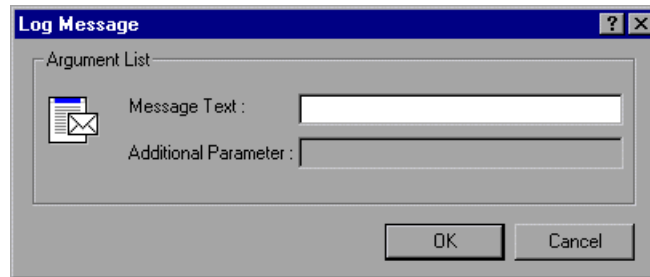
Note: If you modify the script's debugging level using the Log run-time settings, the behavior of the `lr_message`, `lr_output_message`, and `lr_log_message` functions will not change, and they will continue to send messages.

Log Messages

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

To insert an `lr_log_message` function:

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



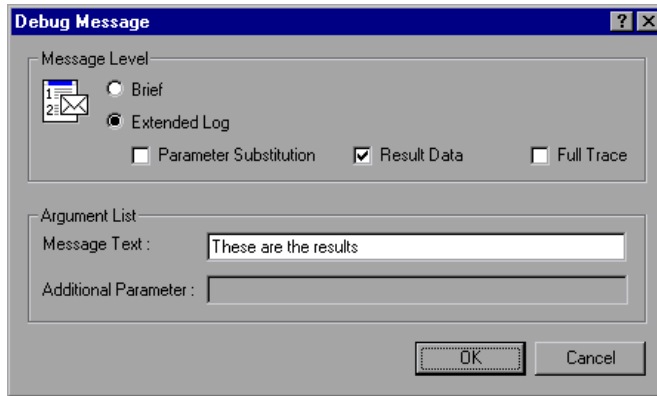
- 2 Type the message into the **Message Text** box.
- 3 Click **OK** to insert the message and close the dialog box. An `lr_log_message` function is inserted at the current point in the script.

Debug Messages

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

To insert a debug function:

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



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

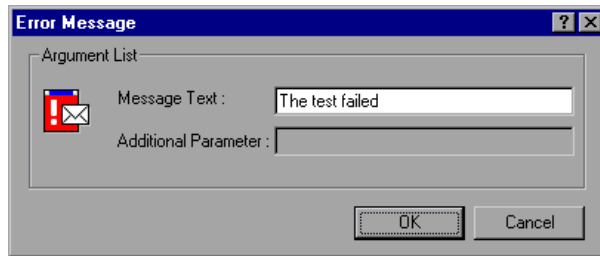
Error and Output Messages

For protocols with a Tree view representation of the script, such as Web, Winsock, and Oracle NCA, you can add an error or output message using the user interface. A common usage of this function is to insert a conditional statement, and issue a message if the error condition is detected.

To insert an error or output message function:

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

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



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

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

Handling Errors in Vuser Scripts During Execution

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

- Using run-time settings. You can specify the **Continue on Error** run-time setting. The **Continue on Error** run-time setting applies to the entire Vuser script. You can use the **lr_continue_on_error** function to override the **Continue on Error** run-time setting for a portion of a script. For details, see “Error Handling,” on page 133.
- Using the **lr_continue_on_error** function. The **lr_continue_on_error** function enables you to control error handling for a specific segment of a Vuser script. To mark the segment, enclose it with **lr_continue_on_error(1);** and **lr_continue_on_error(0);** statements. The new error settings apply to the enclosed Vuser script segment. See the paragraphs below for details.

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

```
web_link("EBOOKS",
        "Text=EBOOKS",
        "Snapshot=t2.inf",
        LAST);

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

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

```
lr_continue_on_error(1);
web_link("EBOOKS",
        "Text=EBOOKS",
        "Snapshot=t2.inf",
        LAST);

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

Synchronizing Vuser Scripts

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

The following is a list of the available synchronization functions:

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

For details on using synchronization functions in RTE Vuser scripts, see Chapter 51, “Synchronizing RTE Vuser Scripts.”

Emulating User Think Time

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

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

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

Handling Command Line Arguments

You can pass values to a Vuser script at run-time by specifying command line arguments when you run the script. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

<code>lr_get_attrib_double</code>	Retrieves double precision floating point type arguments
<code>lr_get_attrib_long</code>	Retrieves long integer type arguments
<code>lr_get_attrib_string</code>	Retrieves character strings

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

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

```
script_name /argument argument_value /argument argument_value
```

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

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

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

Encrypting Text

You can encrypt text within your script to protect your passwords and other confidential text strings. You can perform encryption both automatically, from the user interface, and manually, through programming. When you encrypt a string, it appears in the script as a coded string. In order for the script to use the encrypted string, it must be decrypted with `lr_decrypt`.


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

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

To encrypt a string:

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

To restore an encrypted string:

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

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

Using C Functions in Vuser Scripts

VuGen generates Vuser scripts in C. All standard ANSI C conventions apply to the scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other C programs. You declare and define variables using ANSI C conventions.

The *Online Function Reference* (**Help > Function Reference**) contains a C reference with syntax and examples of commonly used C functions.

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.

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 when necessary, or ask Mercury Interactive 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. */
```

7

Defining Parameters

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

This chapter describes:

- ▶ Understanding Parameter Limitations
- ▶ Creating Parameters
- ▶ Defining Parameter Properties
- ▶ Understanding Parameter Types
- ▶ Assigning Internal Data
- ▶ Specifying a Parameter Format
- ▶ Selecting a File as a Source for Parameter Values
- ▶ Importing Data from Existing Databases
- ▶ User-Defined Functions
- ▶ Parameterization Options

The following information applies to all types of Vuser scripts.

About Defining Parameters

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

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

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

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

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

The resulting Vusers then substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables. For more information about data sources, see “Understanding Parameter Types” on page 83.

Parameterizing a Vuser script has two advantages:

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

Parameterization involves the following two tasks:

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

Understanding Parameter Limitations

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

For example, consider the `lrd_stmt` function. The function has the following syntax:

```
lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long mliTextLen, LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int miDBErrorSeverity);
```

The *Online Function Reference* indicates that you can parameterize only the `mpcText` argument.

A recorded **lrd_stmt** function could look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name = \"Kim\" ", -1, 148, -99999, 0);
```

You could parameterize the recorded function to look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name = \"<name>\" ", -1, 148, -99999, 0);
```

Note: You can use the **lr_eval_string** function to “parameterize” a function argument that you cannot parameterize by using standard parameterization. In addition, you can use the **lr_eval_string** function to “parameterize” any string in a Vuser script. For more information on the **lr_eval_string** function, see the *Online Function Reference*.

Creating Parameters

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

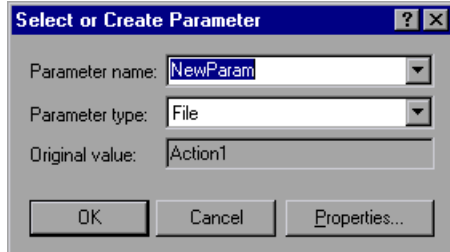
To create a parameter:

- 1 In Script View:** Select a string and select **Replace with a Parameter** from the right-click menu.
- 2 In Tree View:** Select the step you want to parameterize, and select **Properties** from the right-click menu. The appropriate properties dialog box opens.



Click the **ABC** icon that is beside the argument to be parameterized.

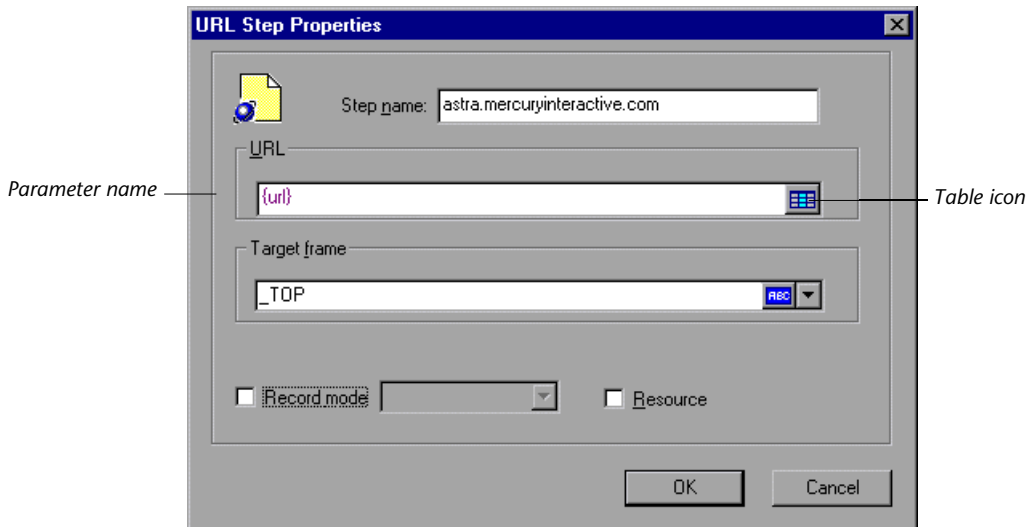
The Select or Create Parameter dialog box opens.



- 3** Type a name for the parameter in the **Parameter name** box, or select an existing parameter name from the list.
- 4** Select a parameter type from the **Parameter type** list. The available types are Date/Time, File, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, User Defined Function, or Vuser ID. For information on the parameter types, see “Understanding Parameter Types” on page 83.
- 5** Click **OK** to close the Select or Create Parameter dialog box. VuGen replaces the selected string in your script with the name of the parameter, surrounded by brackets.



In Tree view, VuGen replaces the ABC icon with the table icon. In the example below, the original URL value, “http://www.merc-int.com/,” has been replaced with the parameter *{url}*.



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

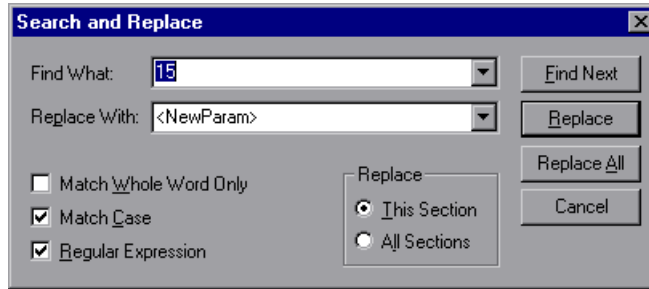
Note: The default parameter braces are either curly or angle brackets, depending on the protocol type. You can change the parameter braces from the Parameterization tab in the General Options dialog box (select **Tools > General Options**). For more information, see “Parameterization Options” on page 105.

- 6 To replace more occurrences of the string with the same parameter, select the parameter and choose **Replace More Occurrences** from the right-click menu. The Search and Replace dialog box opens.

The **Find What** box displays the value you want to replace. The **Replace With** box displays the parameter name in brackets.

Select the appropriate check boxes for matching whole words or case. To search with regular expressions (., !, ?, etc.) select the Regular Expressions check box. For more information, see “Using Regular Expressions,” on page 489.

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



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

- 7** To replace a string with a previously defined parameter, enter Script view. Right-click on the string and select **Use Existing Parameters**. The **Use Existing Parameters** submenu opens.

Select a parameter from the **Use Existing Parameters** submenu, or choose **Select from Parameter List** to open the Parameter List dialog box.

Note: Using the **Parameter List** is convenient when you want to replace a string with a previously defined parameter and, at the same time, view or modify that parameter’s properties. For details on using the Parameter List, see “Using the Parameter List” on page 104.

- 8** To restore a parameter to its original value:

In Script view, right-click on the parameter and select **Restore Original Value**.

In Tree view, right-click on the step and click the table icon. and select **Undo Parameter** from the pop-up menu. The original value is restored.

Defining Parameter Properties

After you create a parameter, you define its properties. A parameter's properties define the data source the parameter uses during script execution.

To define a parameter's properties:

- 1 In Script view:** Select the parameter. Choose **Parameter Properties** from the right-click menu.

In Tree view: Right-click the step containing the parameter whose properties you want to define, and select **Properties**. The appropriate step properties dialog box opens. Click the table icon beside the parameter whose properties you want to define, and select **Parameter Properties** from the pop-up menu.



The Parameter Properties dialog box opens and displays the properties for the current parameter type. In the following example, the properties of a *file* type parameter are displayed.

Parameter Properties - [ServerName]

Parameter type: File

File path: Details.dat Browse...

	ID	FirstName	LastName	Title
1	14	Tom	Smith	Resident
2	23	Jane	WInter	VP Marketin

Edit... Data Wizard... Add Col...

Select column:

By number: 1

By name:

File format:

Column delimiter: Comma

First data line: 1

Select next row: Sequential

Update value on: Each iteration

Close

- 2 Enter the properties of the parameter. For more information, see “Understanding Parameter Types” on page 83.
- 3 Click **Close** to close the Parameter Properties dialog box.

Understanding Parameter Types

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

Assigning Internal Data	Data that is generated internally by the Vuser. This includes Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.
Data Files	Data that is contained in a file—either an existing file or one that you create with VuGen or MS Query.
User-Defined Functions	Data that is generated using a function from an external DLL. For more information about user-defined functions, see “User-Defined Functions” on page 102.

Assigning Internal Data

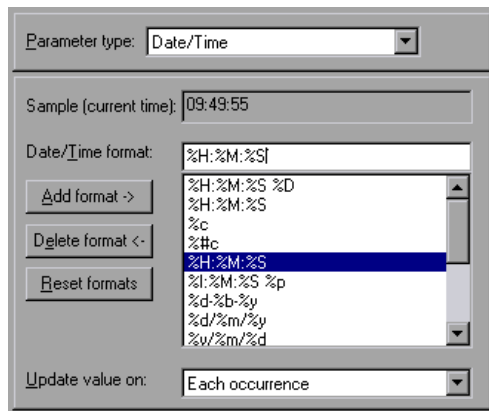
Internal data is generated automatically while a Vuser runs. The following sections describe the various types of internal data.

Date/Time

Date/Time replaces the parameter with the current date and/or time. To specify a date/time format, you can select a format from the menu list or specify your own format. The format should correspond to the date/time format recorded in your script. To create a new format, enter the format in the **Date/Time format** box, and click **Add format**. To delete a format, select it and click **Delete format**. To restore the formats, click **Reset formats**.

The following table describes the date/time symbols:

Symbol	Description
c	complete date and time in digits
#c	complete date as a string and time
H	hours (24 hour clock)
I	hours (12- hour clock)
M	minutes
S	seconds
p	AM or PM
d	date
m	month as a digit (01-12)
b	month as a string - short format (e.g. Dec)
B	month as a string - long format (e.g. December)
y	year in short format (e.g. 03)
Y	year in long format (e.g. 2003)



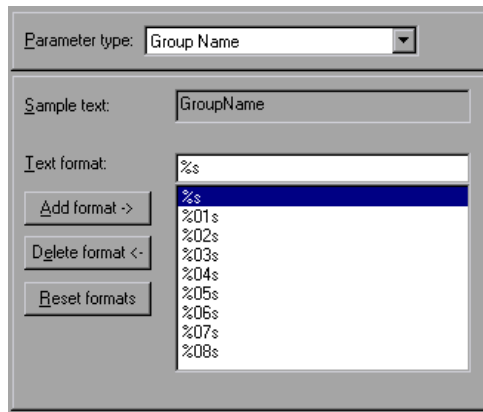
To set the properties for Date/Time type parameters:

- 1 Select one of the existing date/time formats or create a new format. You can view a sample of how VuGen will display the value, in the **Sample Value** box.

- 2 Select an update method, instructing the Vuser when to update parameter values—*Each occurrence*, *Each iteration*, or *Once*. For more information, see “Selecting an Update Method” on page 93.
- 3 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Group Name

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

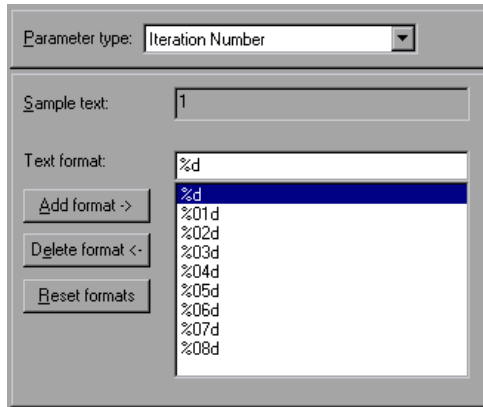


To set properties for the Group Name parameter type:

- 1 Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see “Specifying a Parameter Format” on page 92.
- 2 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Iteration Number

Iteration Number replaces the parameter with the current iteration number.



To set the properties for the Iteration Number parameter type:

- 1** Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see “Specifying a Parameter Format” on page 92.
- 2** Click **Close** to save the settings and close the dialog box.

Load Generator Name

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

The screenshot shows the 'Parameter Properties' dialog box for the 'Load Generator Name' parameter type. The 'Parameter type' dropdown is set to 'Load Generator Name'. The 'Sample text' field contains 'LoadGeneratorName'. The 'Text format' field contains '%s'. Below the 'Text format' field is a list of available formats: '%s', '%01s', '%02s', '%03s', '%04s', '%05s', '%06s', '%07s', and '%08s'. The '%s' format is currently selected. To the left of the list are three buttons: 'Add format ->', 'Delete format <', and 'Reset formats'.

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

- 1** Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see “Specifying a Parameter Format” on page 92.
- 2** Click **Close** to save the settings and close the Parameter Properties dialog box.

Random Number

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

You can use the Random Number parameter type to sample your system's behavior within a possible range of values. For example, to run a query for 50 employees, where employee ID numbers range from 1 through 1000,

create 50 Vusers and set the minimum to 1 and maximum to 1000. Each Vuser receives a random number, from within the range of 1 to 1000.

The screenshot shows the 'Parameter Properties' dialog box for a 'Random Number' parameter type. The 'Parameter type' is set to 'Random Number'. The 'Random range' section has 'Min:' set to '1' and 'Max:' set to '100'. The 'Sample value:' field displays '11'. The 'Number format:' dropdown menu is open, showing options: '%lu', '%03lu', '%04lu', '%05lu', and '%06lu'. The 'Update value on:' dropdown menu is set to 'Each occurrence'.

To set the properties for Random Number type parameters:

- 1** Enter a range defining the set of possible parameter values. You specify minimum and maximum values for the range of random numbers.
- 2** Select a **Number format**, indicating the length of the unique number. Specify `%01lu` for one digit, `%02lu` for two digits, etc. You can view a sample of how VuGen will display the value, in the **Sample Value** box.
- 3** Select an update method, instructing the Vuser when to update parameter values—*Each occurrence*, *Each iteration*, or *Once*. For more information, see “Selecting an Update Method” on page 93.
- 4** Click **Close** to accept the settings and close the Parameter Properties dialog box.

Unique Number

Unique Number replaces the parameter with a unique number. You specify a start number and a block size.

When you use a Unique Number parameter type, you specify a start number and a block size. The block size indicates the size of the block of numbers assigned to each Vuser. Each Vuser begins at the bottom of its range and increments the parameter value for each iteration. For example, if you set

the Start number at 1 with a block of 500, the first Vuser uses the value 1 and the next Vuser uses the value 501, in their first iterations.

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

The image shows a configuration dialog box for a 'Unique Number' parameter. The 'Parameter type' is set to 'Unique Number'. The 'Number range' section includes a 'Start' field with the value '1' and a 'Block size' field with the value '100'. The 'Sample value' field contains '1'. The 'Number format' field is set to '%01d'. The 'Update value on' dropdown is set to 'Each iteration', and the 'When out of' dropdown is set to 'Continue with last value'.

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

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

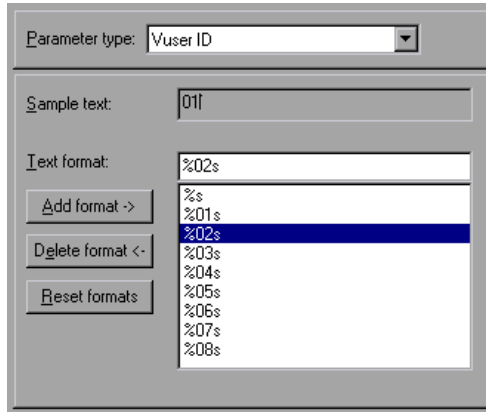
Note: ProTune creates only one instance of Unique Number type parameters. If you define multiple parameters and assign them the Unique Parameter type, the values will not overlap. For example, if you define two parameters with blocks of 100 for 5 iterations, the Vusers in the first group use 1, 101,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

To set the properties for a Unique Number type parameter:

- 1** Enter a start number and the desired block size. For example, if you want 500 numbers beginning with 1, specify 1 in the **Start** box and 500 as a block size.
- 2** Select a Number format, indicating the length of the unique number. Specify %01d for one digit, %02d for two digits, etc. You can view a sample of how VuGen will display the value, in the **Sample Value** box.
- 3** Select an update method, instructing the Vuser when to update parameter values—*Each occurrence*, *Each iteration*, or *Once*. For more information, see “Selecting an Update Method” on page 93.
- 4** Indicate what to do when there are no more unique values, in the **When out of box**: *Abort Vuser*, *Continue in cyclic manner*, or *Continue with last value*.
- 5** Click **Close** to accept the settings and close the Parameter Properties dialog box.

Vuser ID

Vuser ID replaces the parameter with the ID assigned to the Vuser by the Console during a session step run. Note that this is not the ID that appears in the Vuser window—it is a unique ID number generated at runtime. When you run a script from VuGen, the Vuser ID is always -1.



To set the properties for the Vuser ID parameter type:

- 1 Select one of the available formats or create a new one. You select a format to specify the length and structure of the parameter string. For details, see “Specifying a Parameter Format” on page 92.
- 2 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Data Files

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

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

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

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

For details on setting parameter properties for data files, see “Selecting a File as a Source for Parameter Values” on page 94. For details on setting parameter properties for a database, see “Importing Data from Existing Databases,” on page 99.

Specifying a Parameter Format

For most data types, you can specify a format for the parameter by selecting an existing or specifying a new one. You should try to have the parameter format match the recorded values. If the format of the parameter differs from the format of the original recorded value, the script may not run correctly.

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

You can specify a text string before and after the actual parameter value. For example, if you specify a format of “Vuser No: %03s,” a Vuser ID of 1 is displayed as “Vuser No: 001.”

To add a format: Enter the format symbols in the editable box and click **Add Format**. When you add a format to the list, VuGen saves it with the Vuser, making it available for future use.

To delete a format: Select an existing format and click **Delete format**.

To restore the original formats: Click **Reset formats**.

Selecting an Update Method

When using the Date/Time, Random, Unique, and User-Defined Function parameter types, VuGen lets you specify the update method for the parameters. To set the update method, select a method from the **Update value on** list. The available parameter update methods are:

- Each Occurrence
- Each Iteration
- Once

Each Occurrence

The **Each occurrence** method instructs the Vuser to use a new value for each occurrence of the parameter. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter.

Each Iteration

The **Each iteration** method instructs the Vuser to use a new value for each script iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related.

Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. For more information about action blocks, see “Creating Action Blocks,” on page 118.

Once

The **Once** method instructs the Vuser to update the parameter value only once during the session step run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

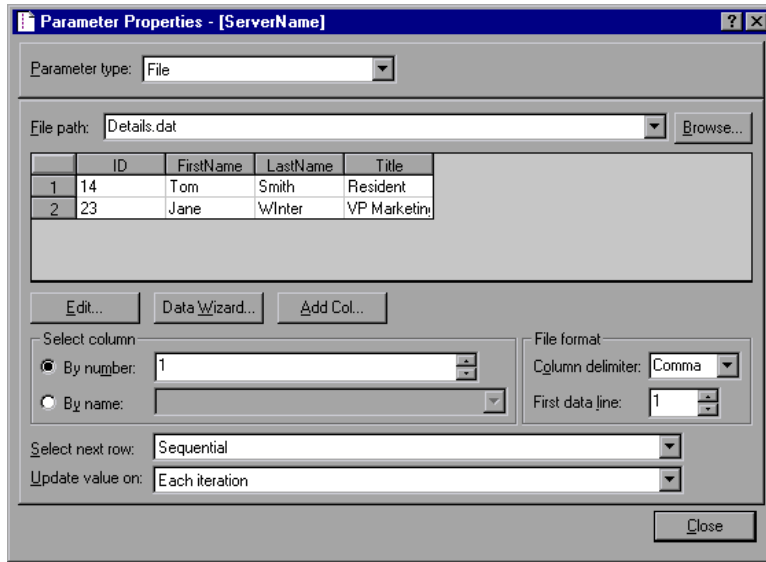
Selecting a File as a Source for Parameter Values

A very common method for using parameters, is instructing Vusers to take values from an external file. Follow these steps:

- Selecting or Creating a Data File
- Setting the Properties for File Type Parameters

Selecting or Creating a Data File

When the parameter type is **File**, the File property settings appear when you open the Parameter Properties dialog box.

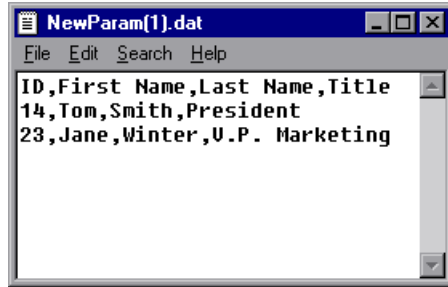


Only the first 100 rows of the data are displayed. To view all of the data, click Edit and view the data in Notepad.

To select a source file for your parameter:

- 1 Type a name for the data file in the **File path** box, or click **Browse** to specify the file location of an existing data file. By default, all new data files are named *parameter_name.dat* and stored in the script's directory. Note that existing data files must have a *.dat* extension.

- Click **Edit**. Notepad opens with the parameter's name in the first row and its original value in the second row. Enter additional column names and values into the file in the form of a table. Use a delimiter such as a comma or a tab to indicate a column break. Begin a new line for each new table row (i.e., for each new row of data).



Note: To add a column to the data file without launching Notepad, click **Add Col** in the Parameter Properties dialog box. The **Add new column** dialog box opens. Type a name for the new column in the **Column name** box, and click **OK**. VuGen adds a new column to the table with the original value of the parameter in row 1.

Setting the Properties for File Type Parameters

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

To set the file type parameter properties:

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

To specify a column number, select **By number** and the column number. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, select 1.

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

- 2** In the **Column delimiter** box of the **File Format** section, enter the column delimiter—the character used to separate the columns in the table. You can specify a comma, tab, or space.
- 3** In the **First data line** box of the **File Format** section, select the first line of data to be used during Vuser script execution. The header is line 0. To begin with the first line after the header, specify 1. If there is no header, specify 0.
- 4** Select an option from the **Select next row** list to instruct the Vuser how to select the table data during Vuser script execution. The options are: Sequential, Random, or Unique. For more information, see “Choosing an Assignment Method for File Type Parameters,” on page 97.
- 5** Choose an update option from the **Update Value on** list. The choices are *Each Iteration*, *Each Occurrence*, and *Once*. For more information, see “Selecting an Update Method,” on page 93.
- 6** If you chose *Unique* as the **Select next row** option:
 - When out of values:** Specify what to do when there is no more unique data: *Abort the Vuser*, *Continue in a cyclic manner*, or *Continue with last value*.
 - Allocate Vuser values in the Controller (Console):** Indicate whether you want to manually allocate data blocks for the Vusers: *Automatically allocate block size* or *Allocate x values for each Vuser*. Specify the number of values to allocate.

Note: You can also set the properties for a **File** parameter type from the Parameter List dialog box. If you are creating a new **File** parameter, VuGen prompts you to create the data file. Click **Create**. An **Edit** button replaces the **Create** button. Click **Edit** to open Notepad and fill in your data, as described above.

Choosing an Assignment Method for File Type Parameters

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

- Sequential
- Random
- Unique

Sequential

The *Sequential* method assigns parameter values to a Vuser sequentially. As a running Vuser accesses the data table, it takes the next available row of data.

If you specify *Each Iteration* in the **Update Value on** list box, the Vuser takes the next value from the data table for each iteration.

If you specify *Each Occurrence* in the **Update Value on** list box, the Vuser takes the next value from the data table for each occurrence of the parameter, even if it is within the same iteration.

If you specify *Once* in the **Update Value on** list box, the value assigned in the first iteration is used for all subsequent iterations for each Vuser.

First Name
Kim
David
Michael
Jane
Ron
Alice
Ken
Julie

For example, assume that your table has the values shown in the table at left.

If you selected *Each Iteration*, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, etc.

If you selected *Each Occurrence*, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, etc.

If you selected *Once*, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, etc.

Random

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

If you specify *Each Iteration* in the **Update Value on** list box, the Vuser takes a new random value from the data table for each iteration.

If you specify *Each Occurrence* in the **Update Value on** list box, the Vuser takes a new random value from the data table for each occurrence of the parameter, even if it is within the same iteration.

If you specify *Once* in the **Update Value on** list box, the random value assigned in the first iteration is used for all iterations of that Vuser.

When running a session step from the ProTune Console, you can specify a seed number for random sequencing. Each seed value represents one sequence of random values used for test execution. Whenever you use this seed value, the same sequence of values is assigned to the Vusers in the session step. You enable this option if you discover a problem in the test execution and want to repeat the test using the same sequence of random values.

For more information refer to the *ProTune Console User's Guide*.

Unique

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

If you specify *Each Iteration* in the **Update Value on** list box, the Vuser takes the next unique value from the data table for each iteration.

If you specify *Once* in the **Update Value on** list box, the unique value assigned in the first iteration is used for all subsequent iterations of the Vuser.

First Name
Kim
David
Michael
Jane
Ron
Alice
Ken
Julie
Fred

For example, assume that your table has the values shown in the table at left.

If you specified *Each Iteration*, for a run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane, Ron, and Alice. The third Vuser, Ken, Julie, and Fred.

If you specified *Once*, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, etc.

Make sure there is enough data in the table for all the Vusers and their iterations. If you have 20 Vusers and you want to perform 5 iterations, your table must contain at least 100 unique values.

If there are not enough values in the data table, you can instruct VuGen how to proceed: *Abort the Vuser*, *Continue in a cyclic manner*, or *Continue with last value*. If you choose to continue with the last value, the Vuser uses the data from the last line of the table for all subsequent iterations.

Suppose you want to allocate values for each Vuser, and you do not want those values shared between Vusers. To accomplish this, you instruct VuGen to allocate a specific number of values for each Vuser in the **Allocate Vuser Values in the Controller** section. By default, VuGen automatically allocates the necessary number of values for the Vusers.

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

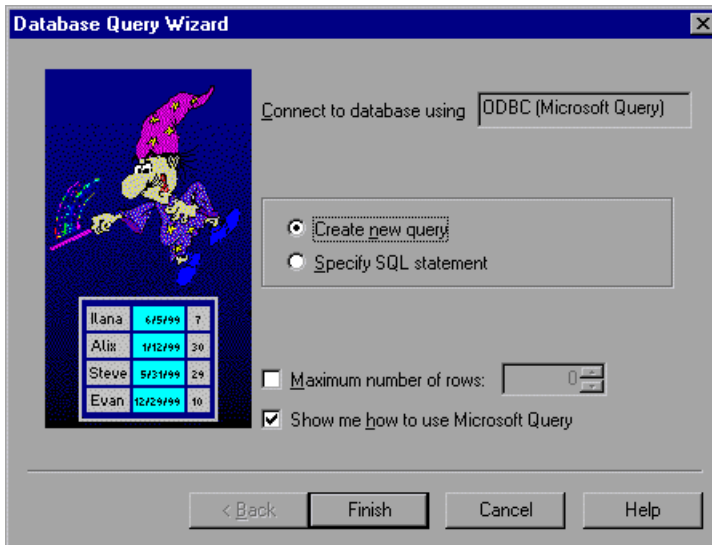
Importing Data from Existing Databases

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

- Creating a New Query
- Specifying an SQL Statement

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

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



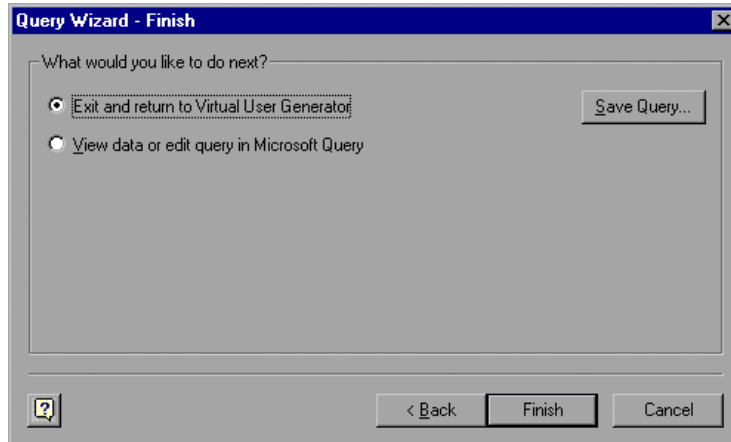
Creating a New Query

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

To create a new query:

- 1 Select **Create new query**. If you need instructions on Microsoft Query, select **Show me how to use Microsoft Query**.
- 2 Click **Finish**. If Microsoft Query is not installed on your machine, VuGen issues a message indicating that it is not available. Install MS Query from Microsoft Office before proceeding.

- 3 Follow the instructions in the wizard, importing the desired tables and columns.
- 4 When you finish importing the data, choose **Exit and return to Virtual User Generator** and click **Finish**. The database records appear in the Parameter Properties box as a data file.



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

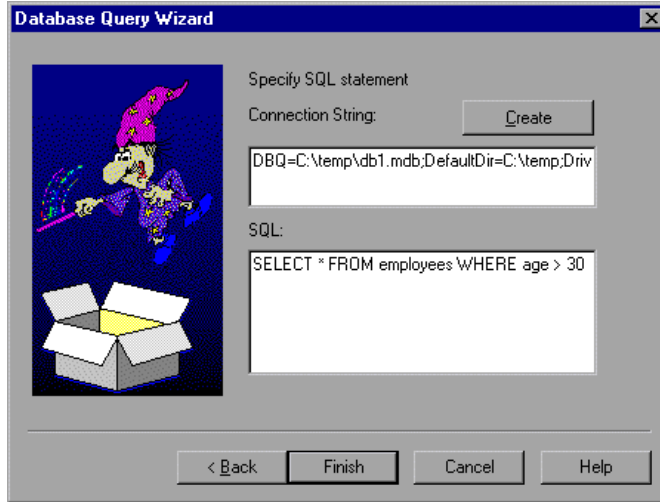
- 5 Set the data assignment properties. See “Setting the Properties for File Type Parameters” on page 95.

Specifying an SQL Statement

To specify a database connection and SQL statement:

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

- 4 In the **SQL** box, type or paste an SQL statement.



- 5 Click **Finish** to process the SQL statement and import the data. The database records appears in the Parameter Properties box as a data file.
- 6 Set the data assignment properties. See “Setting the Properties for File Type Parameters” on page 95.

User-Defined Functions

A user-defined function replaces the parameter with a value returned from a function located in an external DLL.

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

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

The arguments sent to this function are both NULL.

When you create the library, it is recommended that you use the default dynamic library path. That way, you do not have to enter a full path name

for the library, but rather, just the library name. The Virtual User Generator bin directory is on the default dynamic library path. You can add your library to this directory.

The following are examples of user-defined functions:

```
__declspec(dllexport) char *UF_GetVersion( char *x1, char *x2) {return
"Ver2.0";}
```

```
__declspec(dllexport) char *UF_GetCurrentTime(char *x1, char *x2) {
time_t x = tunefully); static char t[35]; strcpy(t, ctime( &x )); t[24] = '\0';
return t;}
```

When you select the **User-Defined Function** type, the user-defined function properties tab opens:

The screenshot shows a dialog box for configuring a user-defined function. At the top, a dropdown menu labeled 'Parameter type:' is set to 'User Defined Function'. Below this is a text field for 'Function Name:' containing 'MyFunctionName'. A section titled 'Library Names:' contains four rows, each with a text field and a 'Browse ...' button. The first row is for 'WinNT Library:' and contains 'MyDll.dll'. The other three rows ('Solaris Library:', 'HP/UX Library:', and 'AIX Library:') are empty. At the bottom, an 'Update value on:' dropdown menu is set to 'Each occurrence'.

To set the properties for user-defined functions:

- 1 Specify the function name in the **Function Name** box. Use the name of the function as it appears in the DLL file.
- 2 In the **Library Names** section, specify a library in the relevant **Library** box. If necessary, locate the file using the **Browse** command.
- 3 Select an update method for the values. For more information on update methods for user-defined functions, see “Selecting an Update Method” on page 93.

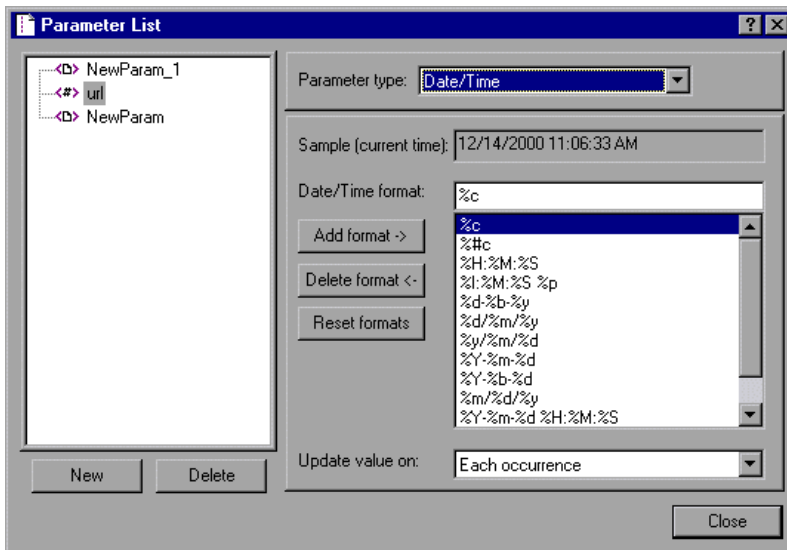
Using the Parameter List

Use the Parameter List to examine all of the parameters, create a new parameter, delete a parameter, or change the properties of an existing parameter.

To use the Parameter List:



- 1 Click the **Parameter List** button or select **Vuser > Parameter List**. Select a parameter to show its properties. In the following example, the properties of a *Date/Time* type parameter are displayed.



- 2 To create a new parameter, click **New**. The new parameter appears in the parameter tree with a temporary name.

Type a name for the new parameter, and press **Enter**.

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

Set the parameter's type and properties, and then click **OK** to close the Parameter List dialog box.

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

- 3** To delete an existing parameter, select the parameter from the parameter tree, click **Delete**, and confirm your action.
- 4** To modify an existing parameter, select the parameter from the parameter tree and edit the parameter's type and properties.

For more information on setting a parameter's properties, see "Understanding Parameter Types" on page 83.

Parameterization Options

You can set the type of braces used to indicate parameters, using the Parameterization tab of the General options.

Parameter Braces

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

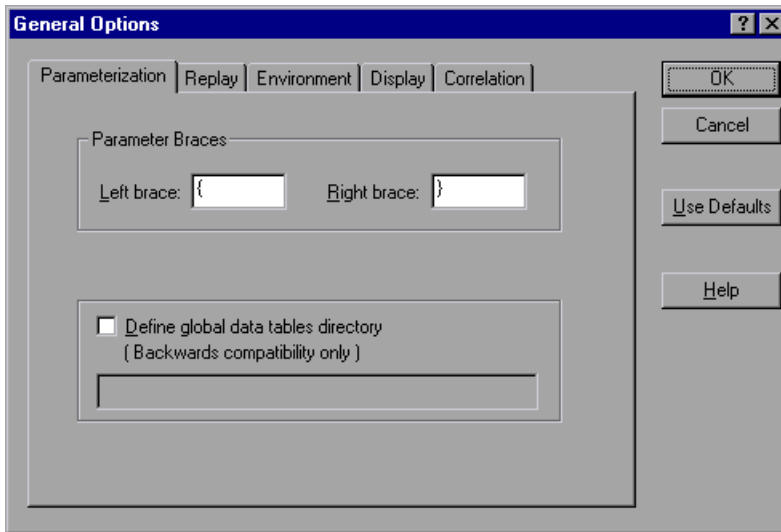
```
web_submit_form("db2net.exe",
  ITEMDATA,
  "name=library.TITLE",
  "value={Book_Title}",
  ENDITEM,
  "name=library.AUTHOR",
  "value=",
  ENDITEM,
  "name=library.SUBJECT",
  "value=",
  ENDITEM,
  LAST);
```

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

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

To change the parameter brace style:

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



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

8

Correlating Statements

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

This chapter describes:

- ▶ Using Correlation Functions for C Vusers
- ▶ Using Correlation Functions for Java Vusers
- ▶ Comparing Vuser Scripts using WDiff
- ▶ Modifying Saved Parameters

The following information applies to all types of Vuser scripts.

About Correlating Statements

The primary reasons for correlating statements are:

- ▶ to simplify or optimize your code

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

- ▶ for dynamic data

Many applications and Web sites identify a session by the current date and time. If you try to replay a script, it will fail, because the current time is

different than the recorded time. Correlating the data enables you to save the dynamic data and use it throughout the session step run.

► to accommodate unique data records

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

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

The main steps in correlating a script are:

1 Determine which value to correlate.

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

Alternatively, you can use the *WDiff* utility distributed with VuGen to determine the inconsistencies within your script. For more information, See “Comparing Vuser Scripts using WDiff” on page 111.

2 Save the results.

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

3 Reference the saved values.

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

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

- For Java language Vusers, see Chapter , “ Correlating Java Scripts.”
- For Database Vusers, see Chapter 18, “Correlating Database Vuser Scripts.”
- For Web Vusers, see Chapter 40, “Configuring Correlation Rules for Web Vuser Scripts.”
- For COM Vusers, see Chapter 25, “Understanding COM Vuser Scripts.”

Using Correlation Functions for C Vusers

To correlate statements for protocols that do not have specific functions, you can use the C Vuser correlation functions. These functions can be used for all C type Vusers, to save a string to a parameter and retrieve it when required. For similar functions for Java, Corba-Java, or RMI-Java Vusers, see “Using Correlation Functions for Java Vusers,” on page 110.

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

For additional information about the syntax of these functions, refer to the *Online Function Reference*.

Using lr_eval_string

In the following example, **lr_eval_string** replaces the parameter *row_cnt* with its current value. This value is sent to the output window using **lr_output_message**.

```
lrd_stmt(Csr1, "select count(*) from employee", -1, 1 /*Deferred*/, ...);
lrd_bind_col(Csr1, 1, &COUNT_D1, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_save_col(Csr1, 1, 1, 0, "row_cnt");
lrd_fetch(Csr1, 1, 1, 0, PrintRow2, 0);
lr_output_message("value : %s", lr_eval_string("The row count is:
<row_cnt>"));
```

Using `lr_save_string`

To save a NULL terminated string to a parameter, use `lr_save_string`. To save a variable length string, use `lr_save_var` and specify the length of the string to save.

In the following example, `lr_save_string` assigns 777 to a parameter `emp_id`. This parameter is then used in another query or for further processing.

```
lrd_stmt(Csr1, "select id from employees where name='John',...");
lrd_bind_col(Csr1,1,&ID_D1,...);
lrd_exec(Csr1, ...);
lrd_fetch(Csr1, 1, ...);
/* GRID showing returned value "777" */
lr_save_string("777", "emp_id");
```

Using Correlation Functions for Java Vusers

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

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

When recording a CORBA-Java or RMI-Java script, VuGen performs correlation internally. For more information see Chapter , “ Correlating Java Scripts.”

Using the Java String Functions

When programming Java Vuser scripts, you can use the Java Vuser string functions to correlate your scripts.

In the following example, `lr.eval_int` substitutes the variable `ID_num` with its value, defined at an earlier point in the script.

```
lr.message(" Track Stock: " + lr.eval_int(ID_num));
```

In the following example, `lr.save_string` assigns John Doe to the parameter `Student`. This parameter is then used in an output message.

```
lr.save_string("John Doe", "Student");
// ...
lr.message("Get report card for " + lr.eval_string("<Student>"));
classroom.getReportCard
```

Comparing Vuser Scripts using WDiff

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

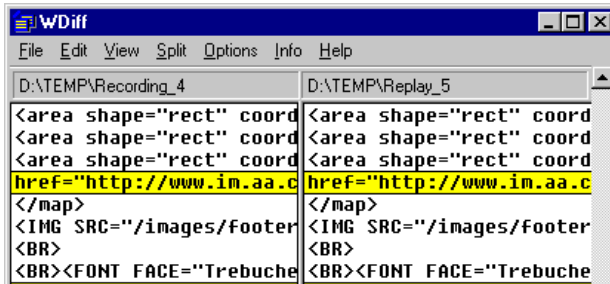
If you are working with other protocols, you can view the Execution log to determine where the script failed and then use the *WDiff* utility to assist you in locating the values that need to be correlated.

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

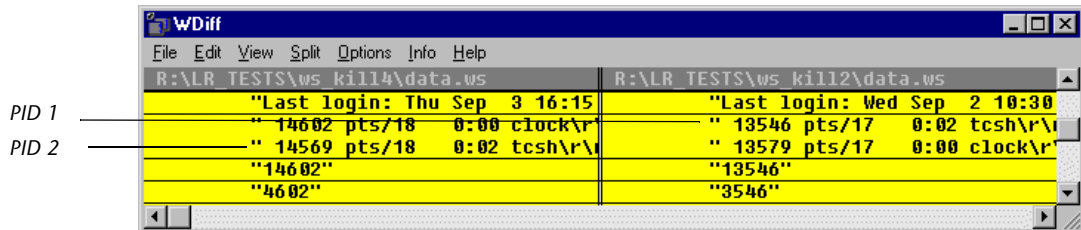
To search for correlations using WDiff:

- 1 Record a script and save it.
- 2 Create a new script and record the identical operations. Save the script.
- 3 Select the section you want to compare (*Actions*, *data.ws*, etc.)

- 4 Select **Tools > Compare with Vuser**. The Open Test box opens.
- 5 Specify a Vuser script for comparison (other than the one in the current VuGen window) and click **OK**. WDiff opens and the differences between the Vuser scripts are highlighted in yellow.



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



- 7 Determine which values need to be correlated.

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

To continue with correlation, refer to the appropriate section:

- For Java language Vusers, see Chapter , “ Correlating Java Scripts.”
- For Database Vusers, see Chapter 18, “Correlating Database Vuser Scripts.”
- For Web Vusers, see Chapter 40, “Configuring Correlation Rules for Web Vuser Scripts.”
- For COM Vusers, see Chapter 25, “Understanding COM Vuser Scripts.”

- For WinSock Vusers, see Chapter 21, “Working with Window Sockets Data.”
- For Tuxedo Vusers, see Chapter 55, “Developing Tuxedo Vuser Scripts.”

Modifying Saved Parameters

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

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

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


9

Configuring Run-Time Settings

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

This chapter describes:

- ▶ Configuring Run Logic Run-Time Settings (multi-action)
- ▶ Pacing Your Actions
- ▶ Configuring Pacing Run-Time Settings (multi-action)
- ▶ Setting Pacing and Run Logic Options (single action)
- ▶ Configuring the Log Run-Time Settings
- ▶ Configuring the Think Time Settings
- ▶ Configuring Additional Attributes Run-Time Settings
- ▶ Configuring Miscellaneous Run-Time Settings
- ▶ Setting the VB Run-Time Settings

The following information applies to all types of Vuser scripts.

About Run-Time Settings

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

Configuring run-time settings allows you to emulate different kinds of user activity. For example, you could emulate a user who responds immediately to output from the server, or a user who stops and thinks before each response. You can also configure the run-time settings to specify how many times the Vuser should repeat its set of actions.

You use the Run-Time Settings dialog box to display and configure the run-time settings tree. You can open these settings in one of the following ways:



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

You can also modify the run-time settings from the ProTune Console

Note: Vuser scripts have individual run-time setting defaults for VuGen and the Console, to support the debugging environment of VuGen and the load testing environment of the Console.

These are the default settings for Vuser scripts in VuGen and the Console:

Think Time: Off in VuGen and replay as recorded in the Console.

Log: Standard in VuGen and off in the Console.

Enable: Loading of Web Resources - On in both VuGen and the Console.

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

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

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

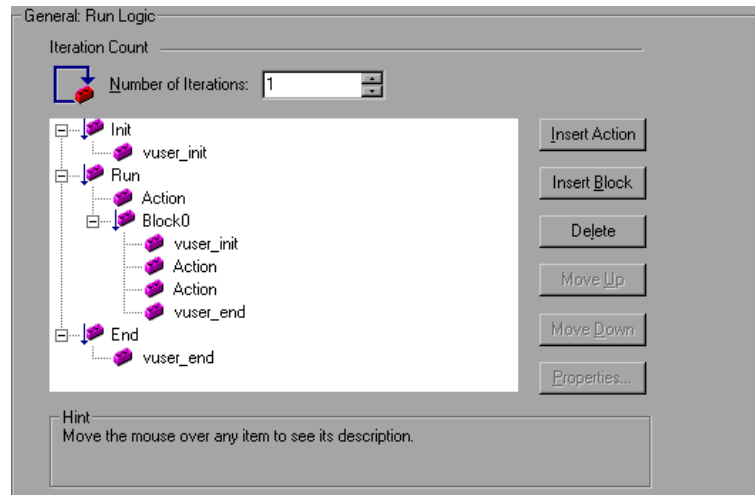
Configuring Run Logic Run-Time Settings (multi-action)

Note: The following section only applies to protocols that support multiple actions. If the first Action section is named **Action1**, multiple actions are supported.

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

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

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



Specify the number of iterations in the **Number of Iterations** box. ProTune repeats all of the Actions the specified number of times.

If you specify a session step duration in the Console's Scheduling settings, the duration setting overrides the Vuser iteration settings. This means that if the duration is set to five minutes (the default setting), the Vusers will continue to run as many iterations as required in five minutes, even if the run-time settings specify only one iteration.

Note: The following section only applies to protocols that support multiple actions. If the first Action section is named **Action1**, multiple actions are supported.

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

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

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

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

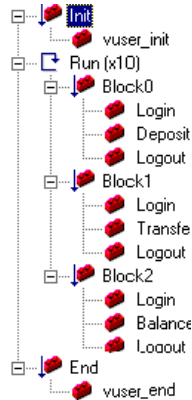
Weighting: For action blocks running their actions randomly, you can set the *weight* or percentage of each action within a block.

Creating Action Blocks

Action blocks are groups of actions within the Vuser script. You can create separate action blocks for groups of actions, adding the same action to several blocks. You can instruct VuGen to execute action blocks or individual actions sequentially or randomly. In the default sequential mode,

the Vuser executes the blocks or actions in the order in which they appear in the iteration tree view.

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

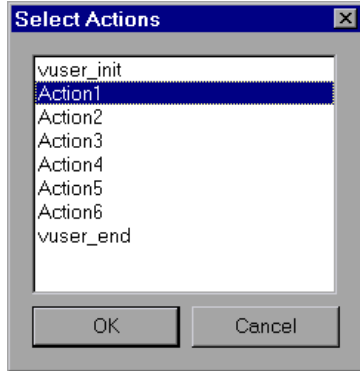


You configure each block independently—its sequence and iterations.

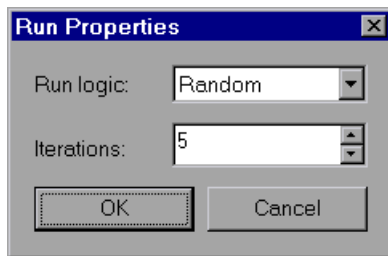
To configure actions and action blocks:

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

- 4 Add actions to the block. Click **Insert Action**. The Select Actions list opens.

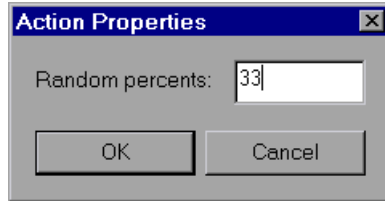


- 5 Select an action to add to the block and click **OK**. VuGen inserts a new action into the current block or section.
- 6 Repeat step 3 for each action you want to add to the block.
- 7 To remove an action or an action block, select it and click **Delete**.
- 8 Click **Move Up** or **Move Down** to modify an item's position.
- 9 Click **Properties** to set the number of iterations and run logic of the actions. The Run Properties dialog opens.



- 10 Select *Sequential* or *Random* from the **Run Logic** list, indicating to VuGen whether to run the actions sequentially or randomly.
- 11 Specify the number of iterations in the **Iterations** box. Note that if you define parameters within the action block, and you instruct VuGen to update their values each iteration, it refers to the global iteration—not the individual block iteration.

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



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

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

Pacing Your Actions

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

- As soon as the previous iteration ends.
- After the previous iteration ends with a fixed/random delay of ...
- At fixed/random intervals, every .../ to ... seconds.

As soon as the previous iteration ends

The new iteration begins as soon as possible after the previous iteration ends.

After the previous iteration ends with a fixed or random delay of ...

Starts each new iteration a specified amount of time after the end of the previous iteration. Specify either an exact number of seconds or a range of time. For example, you can specify to begin a new iteration at any time between 60 and 90 seconds after the previous iteration ends.

The actual amount of time that the Vuser waits between the end of one iteration and the start of the next one appears in the Execution Log when you run the script.

At fixed or random intervals, every ... [to ...] seconds.

You specify the time between iteration—either a fixed number of seconds or a range of seconds from the beginning of the previous iteration. For example, you can specify to begin a new iteration every 30 seconds, or at a random rate ranging from 30 to 45 seconds from the beginning of the previous iteration. Each scheduled iterations will only begin when the previous iteration is complete.

The actual amount of time that the Vuser waits between the end of one iteration and the start of the next one, appears in the Execution Log when you run the script.

Each scheduled iteration will only begin when the previous iteration is complete.

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

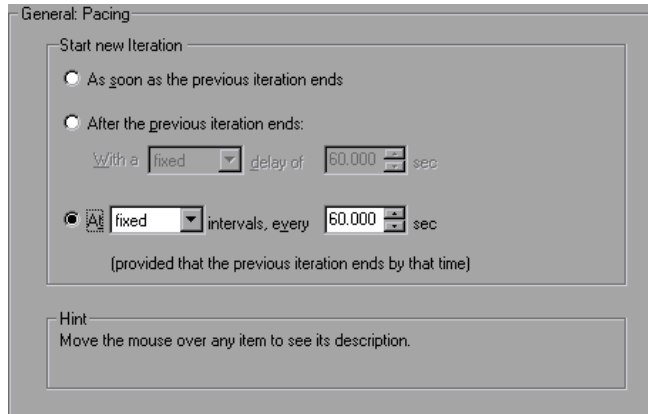
- If the first iteration takes three seconds, the Vuser waits one second.
- If the first iteration takes two seconds to complete, the Vuser waits two seconds.
- If the first iteration takes 8 seconds to complete, the second iteration will start 8 seconds after the first iteration began. ProTune displays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

Configuring Pacing Run-Time Settings (multi-action)

The following section only applies to protocols that work with multiple actions, not a single action. If the **Run Logic** node under the General run-time settings is present, the protocol is supports multi-action.

To set the pacing between iterations:

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



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

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

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

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

- 4 For the **At ... intervals** option:

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

- 5 Click **OK**.

For an overview of the pacing options, see “Pacing Your Actions,” on page 121.

Setting Pacing and Run Logic Options (single action)

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

You can instruct a Vuser to repeat the *Action* section when you run the script. Each repetition is known as an *iteration*. The *vuser_init* and *vuser_end* sections of a Vuser script are not repeated when you run multiple iterations.

To set the iteration and pacing preferences:



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

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

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

- As soon as the previous iteration ends.
 - After the previous iteration ends
 - At fixed or random intervals
- 4 For the **After the previous iteration ends** option:
 - Select a delay type: **fixed** or **random**.
 - Specify a value for fixed, or a range of values for the random delay.
 - 5 For the **At ... intervals** option:
 - Select a interval type: **fixed** or **random**.
 - Specify a value for fixed, or a range of values for the random interval.
 - 6 Click **OK**.

For an overview of the pacing options, see “Pacing Your Actions,” on page 121.

Configuring the Log Run-Time Settings

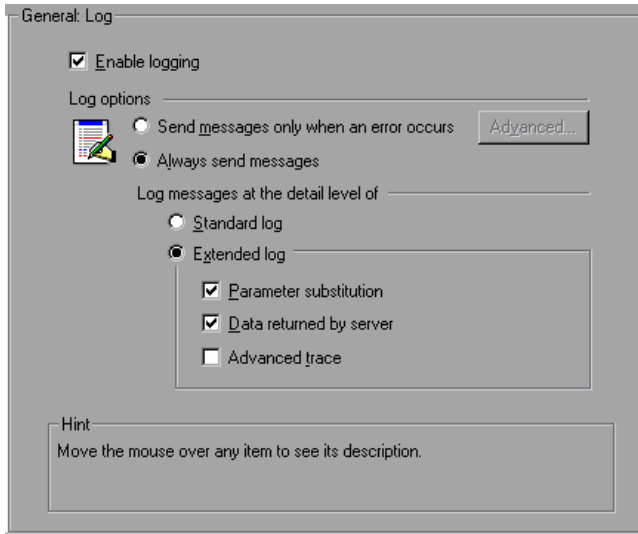
During execution, Vusers log information about themselves and their communication with the server. In a Windows environment, this information is stored in a file called *output.txt* in the script directory. In UNIX environments, the information is directed to the standard output. The log information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

The Log run-time settings let you determine how much information is logged to the output. You can select **Standard** or **Extended** log, or you can disable logging completely. Disabling the log is useful when working with many Vusers. When you create a session step in the Console, logging is automatically disabled. If you have tens or hundreds of Vusers logging their run-time information to disk, the system may work slower than normal. You should only disable logging after verifying that the script is functional.

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



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



Enable Logging

This option enables automatic logging during replay—VuGen writes log messages that you can view in the Execution log. This option only affects automatic logging and log messages issued through `lr_log_message`. Messages sent manually, using `lr_message`, `lr_output_message`, and `lr_error_message`, are still issued.

Log Options

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

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

If you choose to send messages only when errors occur, also known as JIT, (Just in Time) messaging, you can set an additional advanced option,

indicating the size of the log cache. When the contents of the log file exceed the specified size, it deletes the oldest items. The default size is 1KB. Note that in JIT mode, the output of the `lr_message` and `lr_log_message` functions are not sent to the Output window or log file.

Log Detail Level

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

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

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

Standard Log Option

When you select **Standard** log, it creates a standard log of functions and messages sent during script execution to use for debugging. Disable this option for large tuning sessions.

If the logging level is set to *Standard*, the logging mode is automatically set to JIT logging when adding it to a session step.

Extended Log Options

Select **Extended** log to create an extended log, including warnings and other messages. Disable this option for large tuning sessions. If logging is disabled or if the level is set to *Extended*, adding it to a session step does not affect the log settings.

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

- **Parameter substitution:** Select this option to log all parameters assigned to the script along with their values. For more information on parameters, see Chapter 7, “Defining Parameters.”

- ▶ **Data returned by server:** Select this option to log all of the data returned by the server.
- ▶ **Advanced trace:** Select this option to log all of the functions and messages sent by the Vuser during the session. This option is useful when you debug a Vuser script.

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

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

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

If the logging mode was set to **Standard** log, VuGen automatically sets the logging mode to Just-In-Time logging when you copying a script to a session step. If the logging mode was set to **Extended**, or if logging was disabled, copying the script to a session step does not affect its logging settings.

Logging CtLib Server Messages

When you run a CtLib script, (Sybase CtLib, under the Client Server type protocols), all messages generated by the CtLib client are logged in the standard log and in the output file. By default, server messages are not logged. To enable logging of server messages (for debugging purposes), insert the following line into your Vuser script:

```
LRD_CTLIB_DB_SERVER_MSG_LOG;
```

VuGen logs all server messages in the Standard log.

To send the server messages to the Console Output window (in addition to the Standard log), type:

```
LRD_CTLIB_DB_SERVER_MSG_ERR;
```

To return to the default mode of not logging server errors, type the following line into your script:

```
LRD_CTLIB_DB_SERVER_MSG_NONE;
```

Note: Activate server message logging for only a specific block of code within your script, since the generated server messages are long and the logging can slow down your system.

Configuring the Think Time Settings

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

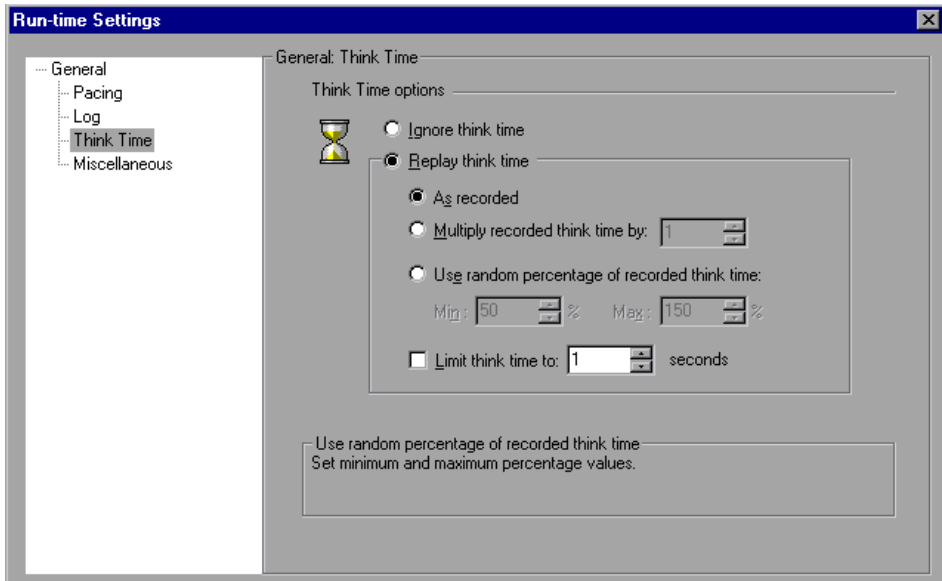
```
lr_think_time(8);
```

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

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



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



Think Time Options

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

Ignore think time: Ignore the recorded think time—replay the script ignoring all `lr_think_time` functions.

Replay the think time

The second set of think times options let you use the recorded think time:

As recorded: During replay, use the argument that appears in the `lr_think_time` function. For example, `lr_think_time(10)` waits ten seconds.

Multiply recorded think time by: During replay, use a multiple of the recorded think time. This can increase or decrease the think time applied during playback. For example, if a think time of four seconds was recorded, you can instruct your Vuser to multiply that value by two, for a total of eight seconds. To reduce the think time to two seconds, multiply the recorded time by 0.5.

Use random percentage of the recorded think time: Use a random percentage of the recorded think time. You set a range for the think time value by specifying a range for the think time. For example, if the think time argument is 4, and you specify a minimum of 50% and a maximum of 150%, the lowest think time can be two (50%) and the highest value six (150%).

Limit think time to: Limit the think time's maximum value.

Configuring Additional Attributes Run-Time Settings

You can use this node to set the Additional Attributes arguments for a Vuser script.

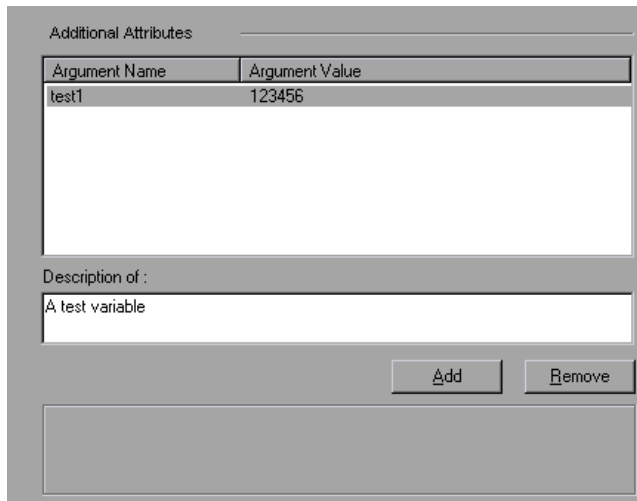
This node lets you specify command line arguments that you can retrieve at a later point during the test run, using `lr_get_attrib_string`.

This allows you to pass in parameters to prepared scripts, enabling you to test and monitor your servers with different client parameters.

- **Add:** Adds a new command line argument entry and enables editing. Enter the argument name and its value.
- **Remove:** Removes the selected argument.



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



The *Additional Attributes* settings apply to all Vuser script types.

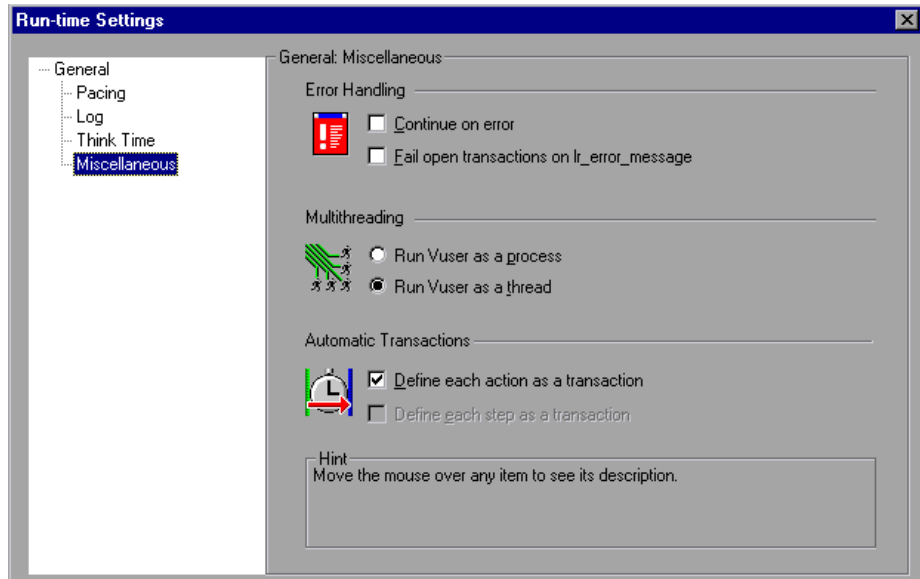
Configuring Miscellaneous Run-Time Settings

You can set the following Miscellaneous run-time options for a Vuser script:

- Error Handling
- Multithreading
- Automatic Transactions



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



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

Error Handling

You can specify how a Vuser handles errors during script execution. By default, when a Vuser detects an error, it exits. You can use the run-time settings to instruct a Vuser to continue script execution when an error occurs. To do so, select the **Continue on Error** check box in the Miscellaneous run-time settings.

You can also instruct VuGen to mark all transactions in which an **lr_error_message** function was issued, as *Failed*. The **lr_error_message** function is issued through a programmed *If* statement, when a certain condition is met.

Error Handling for Database Vusers

When working with database protocols (LRD), you can control error handling for a specific segment of a script. To mark a segment, enclose it

with `LRD_ON_ERROR_CONTINUE` and `LRD_ON_ERROR_EXIT` statements. The Vuser applies the new error setting to the whole segment. If you specify Continue on Error, VuGen issues a messages indicating that it encountered an error and is ignoring it.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
lrd_stmt(Csr1, "select..."...);  
lrd_exec(...);
```

To instruct the Vuser to continue on error for the entire script except for a specific segment, select the **Continue on Error** option and enclose the segment with `LRD_ON_ERROR_EXIT` and `LRD_ON_ERROR_CONTINUE` statements:

```
LRD_ON_ERROR_EXIT;  
lrd_stmt(Csr1, "select..."...);  
lrd_exec(...);  
LRD_ON_ERROR_CONTINUE;
```

In addition to the `LRD_ON_ERROR` statements, you can control error handling using *severity levels*. `LRD_ON_ERROR` statements detect all types of errors—database related, invalid parameters, etc. If you want the Vuser to terminate only when a database operation error occurs (Error Code 2009), you can set a function's severity level. All functions that perform a database operation use severity levels, indicated by the function's final parameter, *miDBErrorSeverity*.

VuGen supports the following severity levels:

Definition	Meaning	Value
LRD_DB_ERROR_SEVERITY_ERROR	Terminate script execution upon database access errors. (default)	0
LRD_DB_ERROR_SEVERITY_WARNING	Continue script execution upon database access errors, but issue a warning.	1

For example, if the following database statement fails (e.g. the table does not exist), the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 0);
```

To instruct VuGen to continue script execution, even when a database operation error occurs, change the statement's severity level from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 1);
```

Note: When you enable Continue on Error, it overrides the “0” severity level; script execution continues even when database errors occur. However, if you disable Continue on Error, but you specify a severity level of “1”, script execution continues when database errors occur.

Error Handling for RTE Users

When working with RTE Users, you can control error handling for specific functions. You insert an `lr_continue_on_error(0);` statement before the function whose behavior you want to change. The User uses the new setting until the end of the script execution or until another `lr_continue_on_error` statement is issued.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
TE_wait_sync();  
TE_type(...);
```

To instruct the Vuser to continue on error for the entire script, except for the following segment, select the **Continue on Error** option and enclose the segment with **lr_continue_on_error** statements, using 0 to turn off Continue on Error and 1 to turn it back on:

```
lr_continue_on_error(0);  
TE_wait_sync();  
lr_continue_on_error(1);  
....
```

Multithreading

Vusers support multithread environments. The primary advantage of a multithread environment is the ability to run more Vusers per load generator. Only threadsafe protocols should be run as threads.

Note: The following protocols are not threadsafe: Sybase-Ctlib, Sybase-Dblib, Informix, Tuxedo, and PeopleSoft-Tuxedo.

- ▶ To enable multithreading, click **Run Vuser as a thread**.
- ▶ To disable multithreading and run each Vuser as a separate process, click **Run Vuser as a process**.

The Console uses a driver program (e.g., mdrv.exe, r3vuser.exe) to run your Vusers. If you run each Vuser as a process, then the same driver program is launched (and loaded) into the memory again and again for every instance of the Vuser. Loading the same driver program into memory uses up large amounts of RAM (random access memory) and other system resources. This limits the numbers of Vusers that can be run on any load generator.

Alternatively, if you run each Vuser as a thread, the Console launches only one instance of the driver program (e.g., mdrv.exe), for every 50 Vusers (by default). This driver process/program launches several Vusers, each Vuser running as a thread. These threaded Vusers share segments of the memory of the parent driver process. This eliminates the need for multiple re-loading of the driver program/process saves much memory space, thereby enabling more Vusers to be run on a single load generator.

Automatic Transactions

You can instruct ProTune to handle every step or action in a Vuser script as a transaction. This is called using automatic transactions. ProTune assigns the step or action name as the name of the transaction. By default, automatic transactions per action are enabled.

Automatic transactions per action can be defined for all protocols.
Automatic transactions per step can be defined only for Web Vusers.

- To disable automatic transactions per action, clear the **Define each action as a transaction** check box. (enabled by default)
- To enable automatic transactions per step, check the **Define each step as a transaction** check box. (disabled by default)

If you disable automatic transactions, you can still insert transactions manually during and after recording. For more information on manually inserting transactions, see Chapter 6, “Enhancing Vuser Scripts.”

Setting the VB Run-Time Settings

Before running your Visual Basic script, you indicate which libraries to reference during replay.

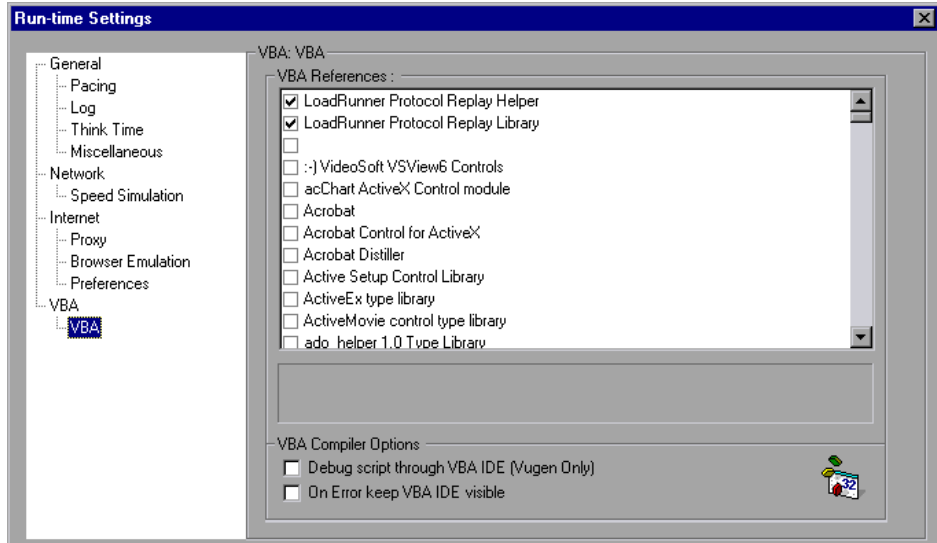


You use the Run-Time Settings dialog box to display and configure the run-time settings. To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar.

You can also modify the run-time settings from the ProTune Console. Click the **Design** tab and click the **Run-Time Settings** button.

To set the VBA Run-Time settings:

- 1 Open the **Run-Time Settings** dialog box and select the **VBA:VBA** node.



- 2 In the **VBA References** section, select the reference library that you want to use while running the script. Select a library to display its description and version in the bottom of the dialog box.
- 3 Select the appropriate compiler options:
Select **Debug script through VBA IDE** to enable debugging through the Visual Basic IDE (Integrated Development Environment).
Select **On Error keep VBA IDE visible** to keep the Visual Basic IDE visible during script execution.
- 4 Choose **OK** to apply the run-time settings.

10

Running Vuser Scripts in Stand-Alone Mode

After you develop a Vuser script and set its run-time settings, you test the Vuser script by running it in stand-alone mode.

This chapter describes:

- ▶ Running a Vuser Script in VuGen
- ▶ Using VuGen's Debugging Features
- ▶ Using VuGen's Debugging Features for Web Vuser Scripts
- ▶ Working with VuGen Windows

The following information applies to all types of Vuser scripts.

About Running Vuser Scripts in Stand-Alone Mode

In order to perform load testing with a Vuser script, you use the Console to incorporate the script into a session step. Before integrating the script into a load testing session step, you check its functionality by running the script in stand-alone mode.

Running a script in stand-alone mode means running the script without using the Console. This is done to establish how the script will execute when run from the Console. To run GUI Vusers in stand-alone mode, use WinRunner. For all other Windows-based scripts, you use VuGen to run scripts in stand-alone mode. If the script is UNIX-based, you run it from a UNIX command line.

When the stand-alone execution is successful, you incorporate the script into a session step. For more information on session steps, refer to your *ProTune Console User's Guide*.

Before you run a script in stand-alone mode, you can:

- ▶ enhance the script with Vuser functions (see Chapter 6, “Enhancing Vuser Scripts”)
- ▶ parameterize the script (see Chapter 7, “Defining Parameters”)
- ▶ correlate queries in the script (see Chapter 8, “Correlating Statements”)

The above steps are optional and may not apply to all scripts.

Running a Vuser Script in VuGen

After developing a Vuser script, run it using VuGen to ensure that it executes correctly.

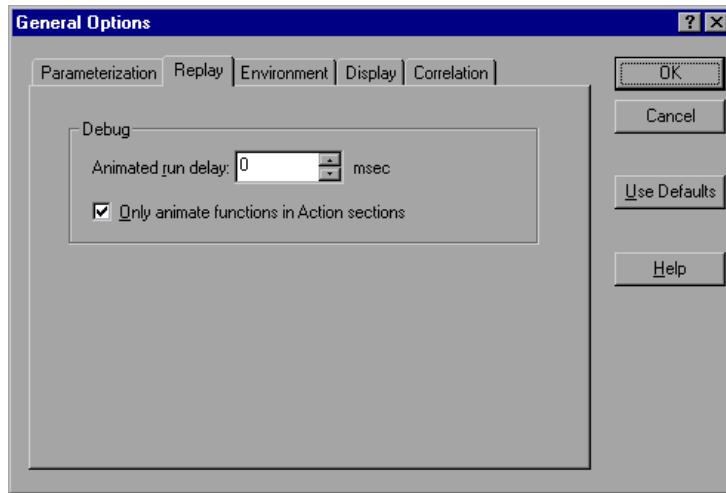
Note: VuGen runs Vuser scripts on Windows platforms only. To run UNIX-based Vuser scripts, see “Running a Vuser Script from a UNIX Command Line,” on page 148.

You can run a Vuser script in animated mode or non-animated mode. When you run in animated mode, VuGen highlights the line of the Vuser script being executed at the current time. You can set a delay for this mode, allowing you to get a better view the effects of each step. When you run in non-animated mode, VuGen executes the Vuser script, but does not indicate the line being executed.

To run a script using VuGen:

- 1 Select **View > Animated Run** to run in animated mode. VuGen places a check mark beside the **Animated Run** menu option to enable animated mode.

- 2 To set the delay for the animated run, select **Tools > General Options**. The General Options dialog box opens.



Select the **Replay** tab. In the **Animated run delay** box, specify a time in milliseconds indicating the delay between commands, and click **OK**. The default delay value is 0 milliseconds. To animate only the content of the Action sections (not in the *init* or *end* sections), select the **Only animate functions in Actions sections** check box.

- 3 If you are running a Web Vuser script, set the Display options (**Tools > General Options**). These options include specifying whether VuGen displays the run-time viewer, whether VuGen generates a report during script execution, and so forth. For more information, see “Using VuGen’s Debugging Features for Web Vuser Scripts” on page 144.
- 4 Select **Vuser > Run**.

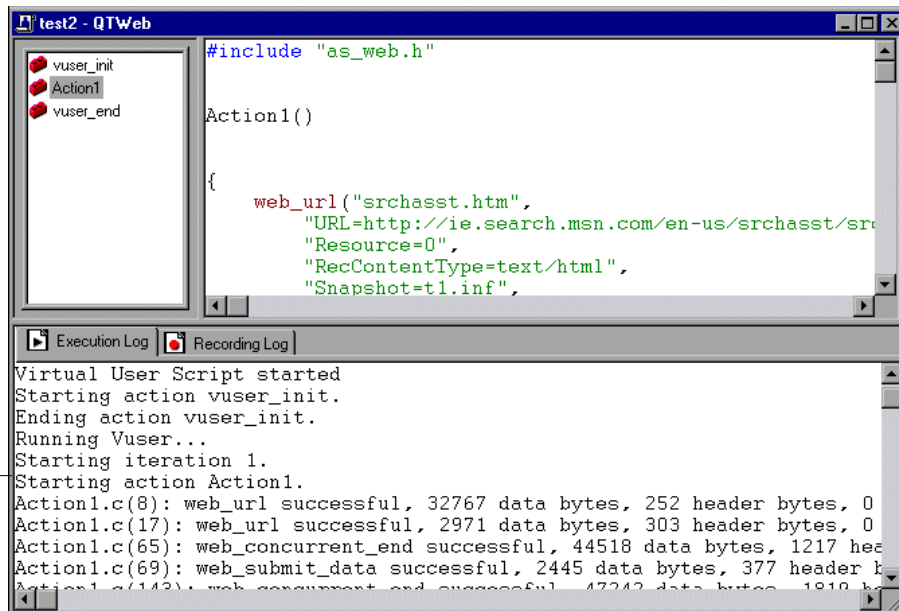
The Output window opens at the bottom of the VuGen main window—or clears, if already open—and VuGen begins executing the Vuser script.

VuGen runs the Vuser script from the first line of the script. The **Execution Log** displays messages that describe the actions of the Vuser as it runs. This information shows you how the script will run when executed in a session step.

Note: For protocols that support tree view (in the **View** menu)—When you run a Vuser script in tree view, VuGen runs the Vuser script from the first icon in the script.

- 5 To hide the Output window during or after a script run, select **View > Output Window**. VuGen closes the Output window and removes the check mark from next to **Output Window** on the **View** menu.
- 6 To interrupt a Vuser script that is running, select **Vuser > Pause**, to temporarily pause the script run, or **Vuser > Stop**, to end the script run.

When script execution is complete, you examine the messages in the execution log to see whether your script ran without errors. The following example shows Execution Log messages from a Web Vuser script run.



Using VuGen's Debugging Features

VuGen contains two options to help debug Vuser scripts—the Run Step by Step command and breakpoints. These options are not available for VBscript and VB Application type Vusers.

VuGen contains additional features to help debug Web Vuser scripts. For details, see “Using VuGen's Debugging Features for Web Vuser Scripts” on page 144.

The Run Step by Step Command

The Run Step by Step Command runs the script one line at a time. This enables you to follow the script execution.

To run the script step by step:



- 1 Select **Vuser > Run Step by Step**, or click the **Step** button.

VuGen executes the first line of the script.

- 2 Continue script execution by clicking the **Step** button until the script run completes.

Breakpoints

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

To set breakpoints:

- 1 Place the cursor on the line in the script at which you want execution to stop.



- 2 Click the **Breakpoint** button. The Breakpoint symbol (👤) appears in the left margin of the script.
- 3 To remove the breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Breakpoint** button.

To run the script with breakpoints:

- 1 Begin running the script as you normally would.

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

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

Once restarted, the script continues until the next breakpoint is encountered or until the script is completed.

Using VuGen's Debugging Features for Web Vuser Scripts

VuGen provides two additional tools to help you debug Web Vuser scripts—the run-time viewer (online browser) and the Results Summary report.

- ▶ You can instruct VuGen to display a run-time viewer when you run a Web Vuser script. The run-time viewer is developed by Mercury Interactive specifically for use with VuGen—it is unrelated to the browser that you use to record your Vuser scripts. The run-time viewer shows each Web page as it is accessed by the Vuser. This is useful when you debug Web Vuser scripts because it allows you to check that the Vuser accesses the correct Web pages. For additional information on the run-time viewer, see Chapter 44, “Power User Tips for Web Vusers.”

Note: To display a run-time viewer you must have Microsoft Internet Explorer 4.0 or higher installed.

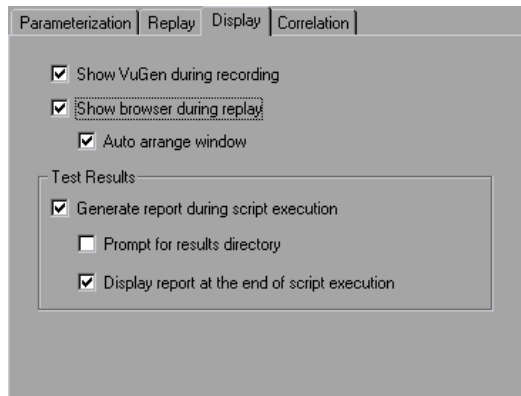
- ▶ You can specify whether or not a Web Vuser generates a Results Summary report during script execution. The Results Summary report summarizes the success or failure of each step in the Web Vuser scripts and allows you to view the Web page returned by each step. For additional details on working with the Results Summary report, see Chapter 43, “Using Reports to Debug Vuser Scripts.”

Note: Transaction times may be increased when a Vuser generates a Results Summary report.

Vusers can generate Results Summary reports only when run from VuGen. When you use the Console to run a Web Vuser script, Vusers cannot generate reports.

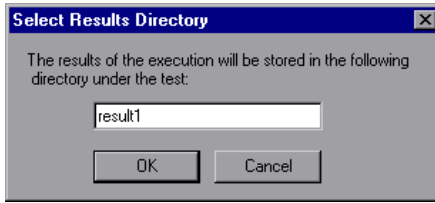
To enable the Web Vuser script debugging features:

- 1** Select **Tools > General Options** from the VuGen menu. The General Options dialog box opens. Select the **Display** tab.



- 2** Select the **Show VuGen during recording** or **Show browser during replay** check boxes to view VuGen during recording or enable the run-time viewer. Select the **Auto arrange window** check box to minimize the run-time viewer when script execution is complete.
- 3** Select the **Generate report during script execution** check box in the **Test Results** section to instruct a Vuser to generate a Results Summary report.

- 4 Select the **Prompt for result directory** check box to display the Select Results Directory dialog box before you execute a Vuser script.



Type a name for the folder where the execution results will be stored, or accept the default name and click **OK**.

If the **Prompt for results directory** check box is not selected, VuGen automatically names the folder *result1*. Subsequent script executions will automatically overwrite previous ones unless a different result file is specified. Note that results are always stored in a subfolder of the script folder.

- 5 Select the **Display report at the end of script execution** check box to automatically display the Results Summary report at the end of script execution. If you do not select this option, you can open the Results Summary report after script execution by selecting **View > Visual Log**.
- 6 For using the Snapshot correlation mechanism, select the **Save correlation information during replay** check box. You can compare each snapshot to your original recorded script.
- 7 Click **OK** to accept the settings and close the General Options dialog box.

Working with VuGen Windows

When you create Vuser scripts, you may need to view several scripts and windows. Use the following VuGen features:

► **Show/Hide the Output Window**

Select **View > Output Window** to show and hide the Output window below the VuGen script editor.

► **Display Grids**

Select **View > Data Grids** to display or hide the grids containing the results data.

► **Close All Windows**

Select **Window > Close All** to close all of the open windows.

Running a Vuser Script from a Command Prompt

You can test a Vuser script from a Command Prompt or from the Windows Run dialog box—without the Console or VuGen user interface.

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

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

```
ProTune VuGen path/bin/mdrv.exe -usr script_name -vugen_win 0
```

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

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

Running a Vuser Script from a UNIX Command Line

When using VuGen to develop UNIX-based Vusers, you must check that the recorded script runs on the UNIX platform. To ensure that your script runs correctly, follow these steps:

1 Test the recorded script from VuGen.

Run the recorded script from VuGen to ensure that the script runs correctly on a Windows-based system.

2 Copy the Vuser script files to a UNIX drive.

Transfer the files to a local UNIX drive.

Note: If you have not already done so, check the Vuser setup on the UNIX machine by using *vu_verify*. For more information about the UNIX Vuser settings, refer to the *Installing ProTune* guide.

3 Test the script from the UNIX command line.

Run the script in stand-alone mode from the Vuser script directory, using the *run_db_vuser* shell script:

```
run_db_vuser.sh script_name.usr
```

Command Line Options: run_db_vuser Shell Script

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

--help

Display the available options. (This option must be preceded by two dashes.)

-cpp_only

Run cpp only (pre-processing) on the script.

-cci_only

Run cci only (pre-compiling) on the script to create a file with a .ci extension. You can run cci only after a successful cpp.

-driver *driver_path*

Use a specific driver program. Each database has its own driver program located in the /bin directory. For example, the driver for CtLib located in the /bin directory, is *mdrv*. This option lets you specify an external driver.

-exec_only

Execute the Vuser .ci file. This option is available only when a valid .ci file exists.

-ci *ci_file_name*

Execute a specific .ci file.

-out *output_path*

Place the results in a specific directory.

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

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

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

Integrating a Vuser Script into a Session Step

Once you have successfully run a script in stand-alone mode to verify that it is functional, you incorporate the script into a session step. A session step contains information about the:

- Users that will run the script
- load generator upon which the script will be executed

You create a session step from the ProTune Console. For more information, refer to the *ProTune Console User's Guide*.

11

Managing Scripts Using TestDirector

VuGen's integration with TestDirector lets you manage ProTune Vuser scripts using TestDirector.

This chapter describes:

- ▶ Connecting to and Disconnecting from TestDirector
- ▶ Opening Scripts from a TestDirector Project
- ▶ Saving Scripts to a TestDirector Project

About Managing Scripts Using TestDirector

VuGen works together with TestDirector, Mercury Interactive's Web-based test management tool. TestDirector provides an efficient method for storing and retrieving Vuser scripts, scenarios, and collecting results. You store scripts in a TestDirector project and organize them into unique groups.

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

For more information on working with TestDirector, refer to the *TestDirector User's Guide*.

Connecting to and Disconnecting from TestDirector

If you are working with both VuGen and TestDirector, VuGen can communicate with your TestDirector project. You can connect or disconnect VuGen from a TestDirector project at any time during the testing process.

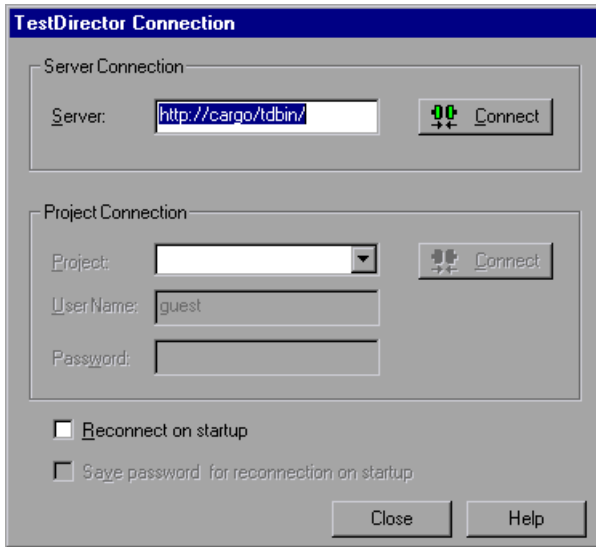
Connecting VuGen to TestDirector

The connection process has two stages. First, you connect VuGen to a local or remote TestDirector Web server. This server handles the connections between VuGen and the TestDirector project.

Next, you choose the project you want VuGen to access. The project stores the scripts for the application you are testing. Note that TestDirector projects are password protected, so you must provide a user name and a password.

To connect VuGen to TestDirector:

- 1 In the VuGen window, choose **Tools > TestDirector Connection**. The TestDirector Connection dialog box opens.



- 2 In the Server box, type the URL address of the Web server on which TestDirector is installed.

Note: You can choose a Web server accessible via a Local Area Network (LAN) or a Wide Area Network (WAN).

- 3 Click **Connect**. Once the connection to the server is established, the server's name is displayed in read-only format in the Server box.
- 4 From the Project box in the Project Connection section, select a TestDirector project.
- 5 In the **User Name** box, type a user name.
- 6 In the **Password** box, type a password.
- 7 Click **Connect** to connect VuGen to the selected project.

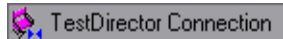
Once the connection to the selected project is established, the project's name is displayed in read-only format in the Project box.

- 8 To automatically reconnect to the TestDirector server and the selected project on startup, select the **Reconnect on startup** check box.
- 9 If you select **Reconnect on startup**, you can save the specified password to reconnect on startup. Select the **Save password for reconnection on startup** check box.

If you do not save your password, you will be prompted to enter it when VuGen connects to TestDirector on startup.

- 10 Click **Close** to close the TestDirector Connection dialog box.

The status bar indicates that VuGen is currently connected to a TestDirector project.

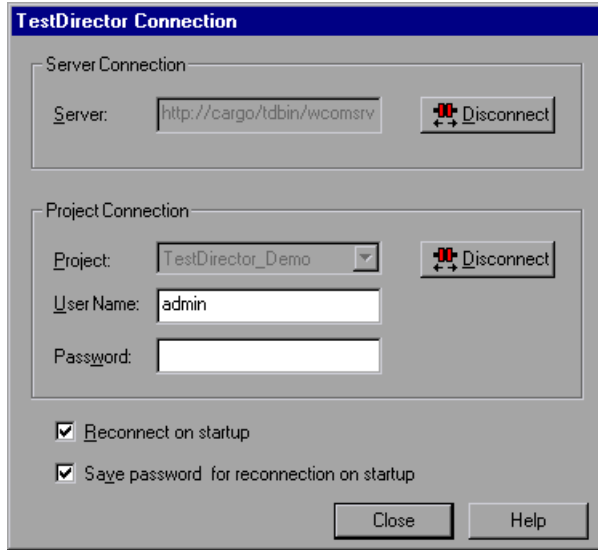


Disconnecting VuGen from TestDirector

You can disconnect VuGen from a selected TestDirector project and Web server.

To disconnect VuGen from TestDirector:

- 1 In the VuGen window, choose **Tools > TestDirector Connection**. The TestDirector Connection dialog box opens.



- 2 To disconnect VuGen from the selected project, click **Disconnect** in the Project Connection section.
- 3 To disconnect VuGen from the selected server, click **Disconnect** in the Server Connection section.
- 4 Click **Close** to close the TestDirector Connection dialog box.

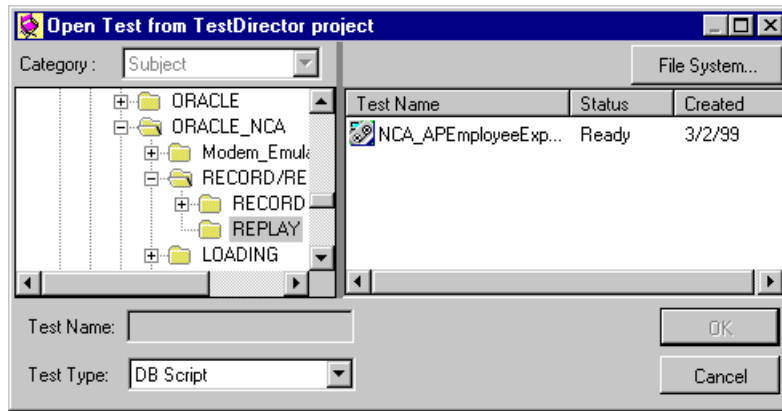
Opening Scripts from a TestDirector Project

When VuGen is connected to a TestDirector project, you can open your scripts from TestDirector. You locate tests according to their position in the test plan tree, rather than by their actual location in the file system.

To open a script from a TestDirector project:

- 1 Connect to the TestDirector server (see “Connecting VuGen to TestDirector” on page 152).

- In VuGen, choose **File > Open** or click the **File Open** button. The Open Test from TestDirector Project dialog box opens and displays the test plan tree.



To open a script directly from the file system, click the **File System** button. The Open Test dialog box opens. (From the Open Test dialog box, you may return to the Open Test from TestDirector Project dialog box by clicking the **TestDirector** button.)

- Click the relevant subject in the test plan tree. To expand the tree and view sublevels, double-click closed folders. To collapse the tree, double-click open folders.

Note that when you select a subject, the scripts that belong to the subject appear in the Test Name list.

- Select a script from the Test Name list. The script appears in the read-only Test Name box.
- Click **OK** to open the script. VuGen loads the script. The name of the script appears in VuGen's title bar. The **Design** tab shows all of the scripts in the test plan tree.

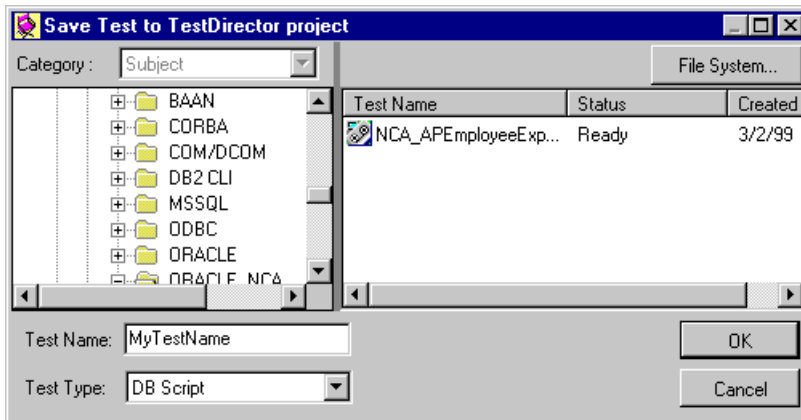
Note: You can also open scripts from the recent file list in the File menu. If you select a script located in a TestDirector project, but VuGen is currently not connected to that project, the TestDirector Connection dialog box opens. Enter your user name and password to log in to the project, and click **OK**.

Saving Scripts to a TestDirector Project

When VuGen is connected to a TestDirector project, you can create new scripts in VuGen and save them directly to your project. To save a script, you give it a descriptive name and associate it with the relevant subject in the test plan tree. This helps you to keep track of the scripts created for each subject and to quickly view the progress of test planning and creation.

To save a script to a TestDirector project:

- 1** Connect to the TestDirector server (see “Connecting VuGen to TestDirector” on page 152).
- 2** In VuGen, choose **File > Save As**. The Save Test to TestDirector Project dialog box opens and displays the test plan tree.



To save a script directly in the file system, click the **File System** button. The Save Test dialog box opens. (From the Save Test dialog box, you may return to the Save Test to TestDirector Project dialog box by clicking the **TestDirector** button.)

- 3** Select the relevant subject in the test plan tree. To expand the tree and view a sublevel, double-click a closed folder. To collapse a sublevel, double-click an open folder.
- 4** In the Test Name box, enter a name for the script. Use a descriptive name that will help you easily identify the script.
- 5** Click **OK** to save the script and close the dialog box.

The next time you start TestDirector, the new script will appear in TestDirector's test plan tree.

Part III

Working with Java Language Protocols

Working with Java Language Protocols refers to RMI-Java, CORBA-Java, EJB, and Jacada types. For each of the mentioned protocols, refer to the appropriate section. This part contains information that applies to all types of Java Users.

12

Recording Java Language Vuser Scripts

VuGen allows you to record applications or applets written in Java, in protocols such as CORBA, RMI, EJB, or Jacada. You can also use VuGen's navigation tool to add any method to your script.

This chapter describes:

- ▶ Getting Started with Recording
- ▶ Understanding Java Language Vuser Scripts
- ▶ Running a Script as Part of a Package
- ▶ Viewing the Java Methods
- ▶ Manually Inserting Java Methods
- ▶ Configuring Script Generation Settings

The following information applies to CORBA-Java, RMI-Java, EJB, and Jacada Vuser scripts.

About Recording Java Language Vuser Scripts

Using VuGen, you can record a Java application or applet. VuGen creates a pure Java script enhanced with ProTune-specific Java 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 ProTune session step.

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. Please refer to Chapter 23, “Programming Java Scripts” for important information about function syntax and system configuration.

Note that 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 ProTune.

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 402

Getting Started with Recording

The following procedure outlines how to record Java language Vuser scripts.

1 Ensure 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 23, “Programming Java Scripts” and the Read Me file.

2 Create a new Vuser script.

Select a protocol type (Distributed Components, EJB, or Middleware) and choose the desired Vuser type.

3 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. For more information, see Chapter 13, “Setting Java Recording Options.”

4 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.

5 Enhance the Vuser script.

Add ProTune specific functions to enhance the Vuser script. For details, see Chapter 23, “Programming Java Scripts.” You can use the built-in Java function Navigator. For more information, see “Viewing the Java Methods” on page 165.

6 Parameterize the Vuser script.

Replace recorded constants with parameters. You can parameterize complete strings or parts of a string. Note that you can define more than one parameter for functions with multiple arguments. For details, see Chapter 7, “Defining Parameters.”

7 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 15, “Configuring Java Run-Time Settings.”

8 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

Refer to the chapter for each Vuser type, for detailed information on the recording procedure.

Understanding Java Language Vuser Scripts

When you record a session, VuGen logs all calls to the server and generates a script with ProTune enhancements. 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 ProTune 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 327.

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.

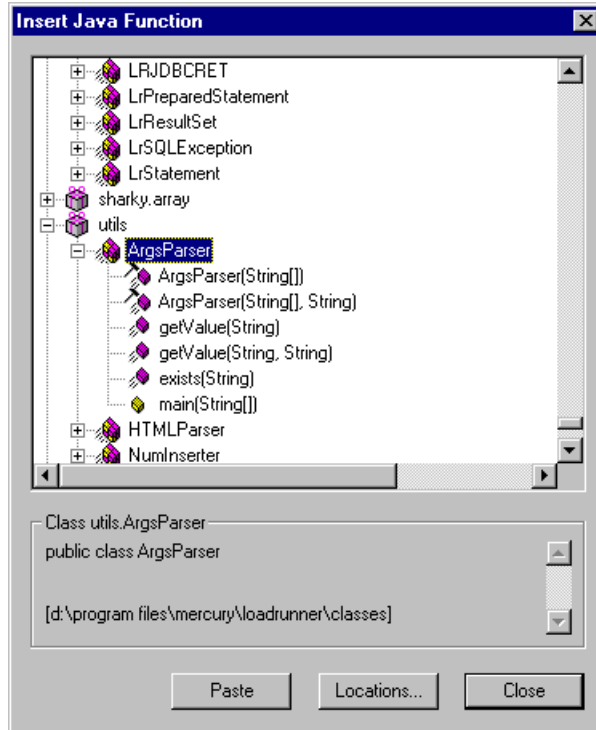
To use the protected methods, add the Vuser to the package of required methods. At the beginning of your script, add the following line:

```
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 166.

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 ()

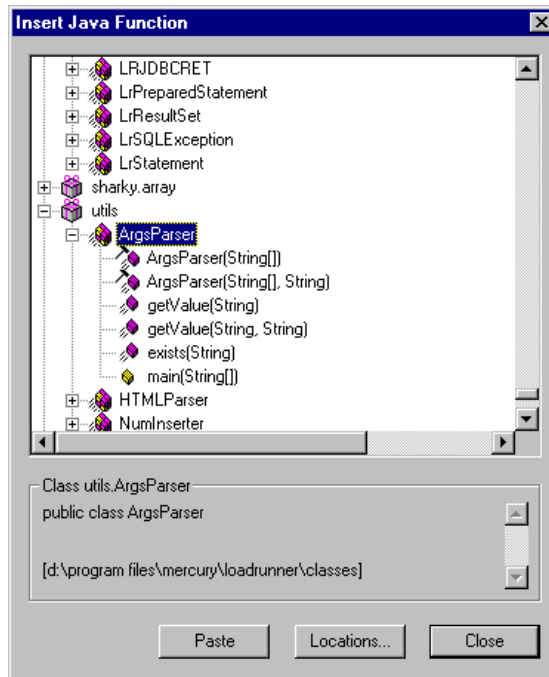
Manually Inserting Java Methods

You use the Java Function navigator to view and add Java functions to your script. The following section apply to EJB Testing, RMI-Java, and CORBA-Java Vusers. You can customize the function generation settings by modifying the configuration file. For more information, see “Configuring Script Generation Settings,” on page 169.

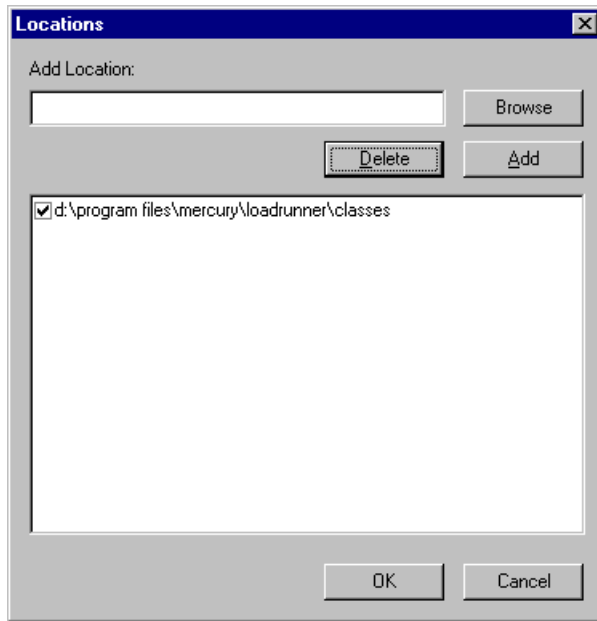
To insert Java functions:

- 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 Choose **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, choose **Browse > Folder**. To add an archive (*jar* or *zip*), choose **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 settings **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 ProTune Java functions (described in Chapter 23, “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.

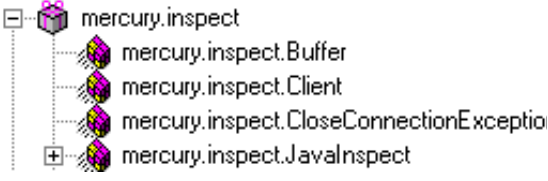

```
[Display]
FullClassName=False

[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

Class Name path

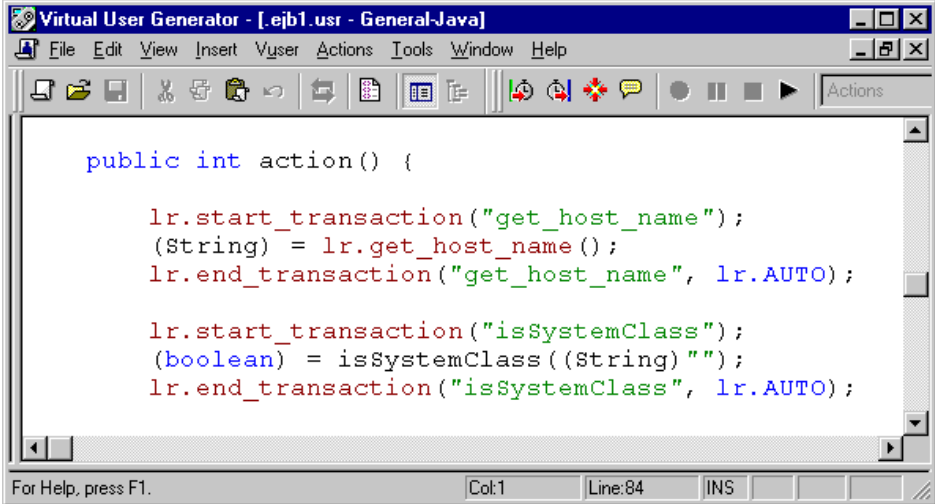
The **FullClassName** option displays the complete package and class name in the Java Function navigator. This option does not affect the way the

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, then you should enable this option.

FullClassName enabled	FullClassName disabled
	

Automatic Transactions

The **AutoTransaction** setting creates a ProTune transaction for all methods. When you enable this option, VuGen automatically encloses all Java methods with `lr.start_transaction` and `lr.end_transaction` functions. This allows you to individually track the performance of each method. This option is disabled by default.



```

public int action() {

    lr.start_transaction("get_host_name");
    (String) = lr.get_host_name();
    lr.end_transaction("get_host_name", lr.AUTO);

    lr.start_transaction("isSystemClass");
    (boolean) = isSystemClass((String) "");
    lr.end_transaction("isSystemClass", lr.AUTO);
}
    
```

Default Parameter Values

The **DefaultValues** setting includes default values for all methods you paste into your script. This option is enabled by default and inserts a null for all objects. If you disable this option, you must manually insert parameter values for all functions in the script. The following table illustrates the **DefaultValues** flag enabled and disabled.

DefaultValues enabled	DefaultValues disabled
lr.message((String)"); lr.think_time((int)0); lr.enable_redirection((boolean>false); lr.save_data((byte[])null, (String)");	lr.message((String)); lr.think_time((int)); lr.enable_redirection((boolean)); lr.save_data((byte[]), (String));

Class Pasting

The **CleanClassPaste** setting pastes a class so that it will compile cleanly: with an instance returning from the constructor, with default values as parameters, and without a need for import statements. Using this option, you will most likely be able to run your script without any further modifications. If you disable this option (default), you may need to manually define parameters and include import statements. Note that this setting is only effective when you paste an entire class into your script—not when you paste a single method.

The following segment shows the *toString* method pasted into the script with the **CleanClassPaste** option enabled.

```
_class.toString();  
    // Returns: java.lang.String
```

The same method with the **CleanClassPaste** option disabled is pasted as follows:

```
(String) = toString();
```

The next segment shows the *NumInserter* Constructor method pasted into the script with the **CleanClassPaste** option enabled.

```
utils.NumInserter _numinsserter = 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)"",...);
```

13

Setting Java Recording Options

VuGen allows you to control the way in which you record your CORBA, RMI, or EJB application. You can use the default recording options, or customize them for your specific needs.

This chapter describes:

- ▶ Java Virtual Machine (JVM) Options
- ▶ Recorder Options
- ▶ Serialization Options
- ▶ Correlation Options
- ▶ Debug Options

The following information applies to CORBA-Java, RMI-Java, and EJB Vuser scripts.

About Setting Java Recording Options

Using VuGen, you record a CORBA (Common Object Request Broker Architecture) or RMI (Remote Method Invocation) Java application or applet. For recording an EJB test, see Chapter 45, “Performing EJB.”

Before recording, VuGen lets you set recording options for the Java Virtual Machine (JVM) and for the code generation stage. Setting the recording options is not mandatory; if you do not set them, VuGen uses the default values.

The options described in this chapter were previously handled by modifying the *mercury.properties* file.

You can set recording options in the following areas:

- Java Virtual Machine (JVM) Options
- Classpath Options
- Recorder Options
- Serialization Options
- Correlation Options
- Debug Options

Java Virtual Machine (JVM) Options

The **Java VM** options indicate additional parameters to use when recording Java applications.

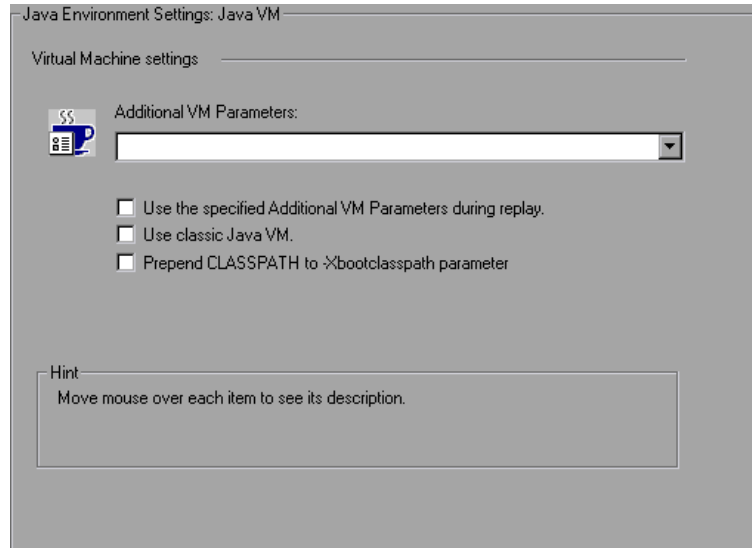
When you record a Vuser, VuGen automatically sets the *Xbootclasspath* variable with default parameters. If you use this dialog box to set the *Xbootclasspath* with different parameters, it will use those command parameters—not the default ones.

You can also instruct VuGen to add the Classpath before the *Xbootclasspath* (prepend the string) to create single Classpath string.

By default, VuGen uses the classic VM during recording. You can also instruct VuGen to use another virtual machine (Sun's Java Hotspot VM).

To set the Java Virtual Machine recording options:

- 1 Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Java VM** node in the Recording Options tree.



- 2 In the **Additional VM Parameters** box, list the Java command line parameters. These parameters may be any Java VM argument. The common arguments are the debug flag (*-verbose*) or memory settings (*-ms*, *-mx*). For more information about the Java VM flags, see the JVM documentation. In addition, you may also pass properties to Java applications in the form of a *-D* flag.

VuGen automatically sets the *-Xbootclasspath* variable (for JDK 1.2 and higher) with default parameters. When you specify *-Xbootclasspath* with parameter values as an additional parameter, VuGen uses this setting instead of the default one.

- 3 To use the same Additional VM parameters in replay, select the **Use the specified Additional VM Parameters during replay** check box.
- 4 To use the classic VM, select the **Use classic Java VM** check box (default). To use another VM, (Sun's Java HotSpot) clear the check box.
- 5 To add the Classpath before the *Xbootclasspath* (prepend the string), select the **Prepend CLASSPATH to -Xbootclasspath parameter** check box.

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

Classpath Options

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper recording.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.

To set the Classpath recording options:

- 1 Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Classpath** node in the Recording Options tree.

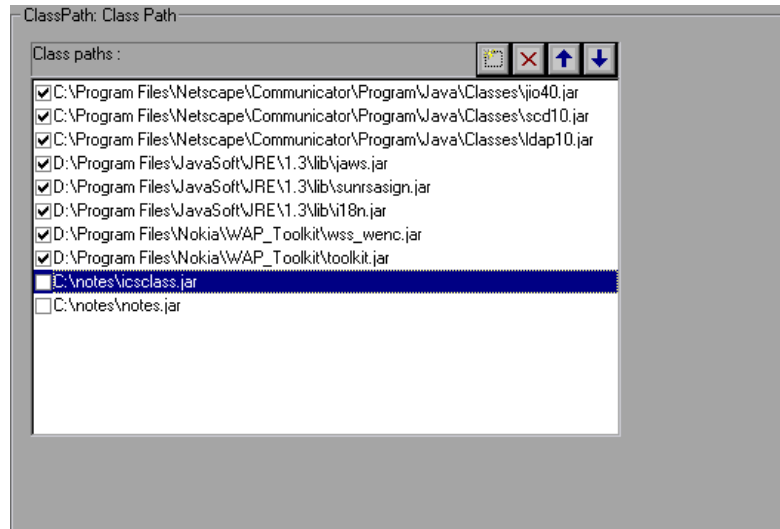





- 2 To add a classpath to the list:

Click the Add Classpath button. VuGen adds a new line to the classpath list.

Type in the path and name of the *jar*, *zip* or other archive file for your class. Alternatively, click the Browse button to the right of the field, and locate the

desired file. VuGen adds the new location to the classpath list, with an enabled status.



-  **3** To permanently remove an entry, select it and click the Delete button.
- 4** To disable a classpath entry for a specific test, clear the check box to the left of the entry.
-  **5** To move an entry down in the list, select it and click the Down arrow.
-  **6** To move a classpath entry up within the list, select it and click the Up arrow.
- 7** Click **OK** to close the dialog box and begin recording.

Recorder Options

The **Recorder** options provide guidelines to VuGen for generating a Vuser script. You can set options in the following areas:

- ▶ General Options
- ▶ Recording Log
- ▶ Styling Options
- ▶ Byte Formatting Options

General Options

Record Return Value: Generates a comment in the script indicating the return value for each invocation. (disabled by default)

Record Progress Messages: Records an `lr.log_message` function before each invocation to allow you to follow the replay progress. (disabled by default)

Record Think Time: Records think times and includes think time function, `lr.think_time`, in the script. (enabled by default)

Record Exception Handling: When an exception occurs, wrap the invocation with a "try-catch" block. (enabled by default)

Insert Functional Check: Inserts verification code that compares the return value received during replay, to the expected return value generated during recording. This option only applies to primitive return values. (disabled by default)

Use LR- API: Includes LR API functions in the script. If you expect to use the script outside of VuGen, disable this option to remove all LR API functions such as think time and other constants. (enabled by default)

Output Redirection: Redirects the *Stdout* and *Stderr* outputs of Java applications to a file. (disabled by default)

Extensions List: A list of all the supported extensions. Each extension has its own hook file. To specify additional extensions, add them to the list of default extensions. If you add extensions to the list, make sure its hook file is available to the Vuser script. The default extensions are JNDI, JMS, and EJB.

Use _JAVA_OPTION flag: Forces JVM versions 1.2 and higher to use the `_JAVA_OPTION` environment variable which contains the desired JVM parameters. (disabled by default)

Recording Log

Generate Recording Log: Generates a recording log displayed in the Output window's Recording tab. If you disable this option, your performance may improve, but no information will be sent to the Output window during recording. (enabled by default)

Generate Variables Info: Writes the inner values of variables to the recording log. If you enable this option, your performance may decrease. (disabled by default)

Detail Level: The number of array elements to show in the log, when recording an array type parameter or return value. The default level is 5.

Styling Options

Use Block Semantics: Places each invocation in a separate scope by wrapping it with curled brackets. If this option is disabled, the entire Action method is wrapped with curled brackets—not each invocation. (disabled by default)

Underscored Variable Names: Precedes all variables generated in the script with an underscore prefix. This is necessary to prevent conflicts with a package of the same name. (enabled by default)

Max Line Length: The maximum length of a recorded line. If any recorded line exceeds this value, it is truncated. VuGen applies a smart truncation, in order not to break any code consistency such as quotes or function parameters. The default value is 1000 characters. The maximum length is 30000 characters.

Max Action Length: The maximum size of an action method. The default value is 3000 characters. If an action method exceeds this value, VuGen breaks it up into smaller action methods.

Comment Lines Containing: Comment out all lines in the script containing one of the specified strings. To specify multiple strings, separate the entries with commas. By default, any line with a string containing <undefined>, will be commented out.

Remove Lines Containing: Remove all lines containing one of the specified strings from the script. To specify multiple strings, separate the entries with commas. This feature is useful for customizing the script for a specific tuning goal.

Byte Formatting Options

Bytes as Characters: Displays readable characters as characters with the necessary casting—not in byte or hexadecimal form. (enabled by default)

Implicit Casting: Instructs VuGen to automatically apply casting to all invocations. When you enable this option, casting is not added to the recorded invocations—the compiler handles it implicitly. If you disable this option, VuGen adds casting to the invocations, resulting in a longer script. (false by default)

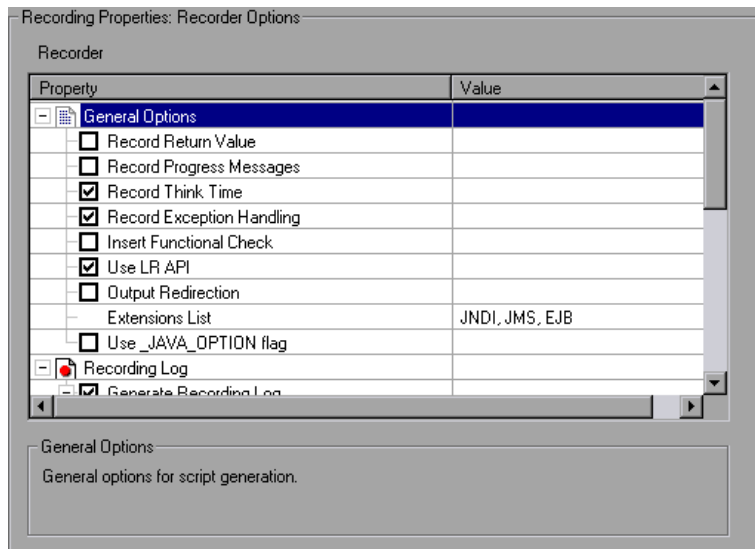
Unreadable Strings as Bytes: Represents strings containing unreadable characters as byte arrays. This option applies to strings that are passed as parameters to invocations. (true by default)

Byte Array Format: The format of byte arrays in a script: *Regular*, *Unfolded Serialized Objects*, or *Folded Serialized Objects*. Use one of the serialized object options when recording very long byte arrays. The default is regular.

Ignore System Properties: Filters out the specified system properties when recording the EJB properties.

To set the Java Recorder options:

- 1 Click **Options** in the Start Recording dialog box and select the **Recording Properties:Recorder Options** node.



- 2 Set the options as desired. For the options with check boxes, select or clear the check box adjacent to the option. For options that require numbers or strings, type in the desired value.
- 3 To set all options to their default values, click **Use Defaults**.
- 4 Click **OK** to close the dialog box and begin recording.

Serialization Options

The **Serialization** options allow you to control how elements are serialized. For an overview of serialization, see “Using the Serialization Mechanism,” on page 194. The following options are available:

Unfold Serialized Objects: Expands serialized objects in ASCII representation. This option allows you to view the ASCII values of the objects in order to perform parameterization. (enabled by default)

Limit Object Size (bytes): Limits serializable objects to the specified value. Objects whose size exceeds this value, will not be given ASCII representation in the script. The default value is 3072.

Serialization Delimiter: Indicates the delimiter separating the elements in the ASCII representation of objects. VuGen will only parameterize strings contained within these delimiters. The default delimiter is ‘#’.

Unfold Arrays: Expands array elements of serialized objects in ASCII representation. If you disable this option and an object contains an array, the object will not be expanded. By default, this option is enabled—all deserialized objects are totally unfolded.

Limit Array Entries: Instructs the recorder not to open arrays with more than the specified number of elements. The default value is 200.

Enable Stub Serialization: Serializes stub objects that were not correlated which would otherwise be <undefined>. Replaying this code on a new server context, may require re-recording. (disabled by default)

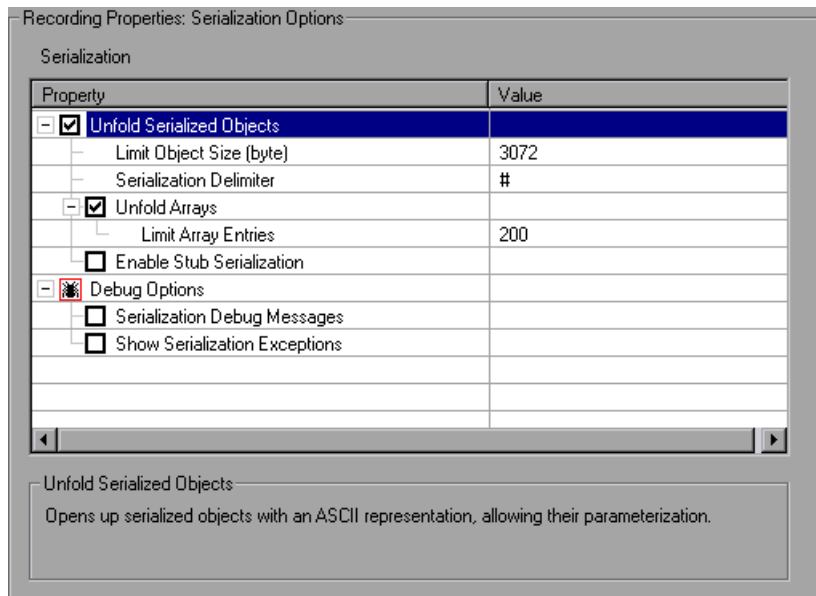
Debug Options

Serialization Debug Messages: Gives debug printouts from serialization mechanism. (disabled by default)

Show Serialization Exceptions: Show all serialization exceptions in the log. (disabled by default)

To set the Serialization options:

- 1 Click **Options** in the Start Recording dialog box and select the **Recording Properties:Serialization Options** node.



- 2 Set the options as desired.
- 3 To set all options to their default values, click **Use Defaults**.
- 4 Click **OK** to close the dialog box and begin recording.

Correlation Options

The **Correlation** options let you indicate whether VuGen should perform automatic correlation, and control its depth. For information about correlation, see Chapter , “Correlating Java Scripts.” The following options are available:

Correlate Strings: Correlate all strings that require correlation. If this option is disabled, VuGen prints them in the script wrapped in quotes. (disabled by default).

Correlate String Arrays: Correlate text within string arrays (enabled by default).

Advanced Correlation: Enables deep correlation in CORBA container constructs and arrays. (enabled by default).

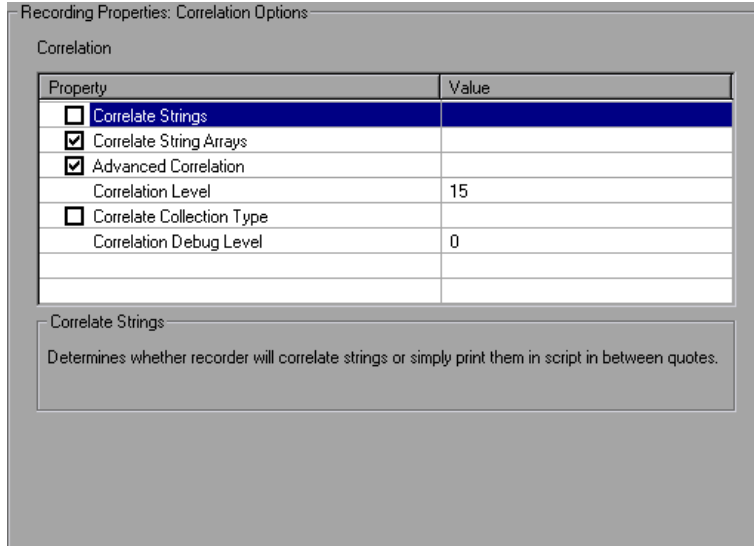
Correlation Level: Indicates the level of deep correlation, the number of inner containers to be scanned. (15 by default).

Correlate Collection Type: Correlates objects from the Collection class for JDK 1.2 and higher. (disabled by default).

Correlation Debug Level: Sends correlation related debug information to the log. You specify a value from 0 through 5. (0 by default, implying no correlation debug information)

To set the correlation options:

- 1 Click **Options** in the Start Recording dialog box and select the **Recording Properties:Correlation Options** node.



- 2 Enable the desired options, or for options that require values, enter the desired value.
- 3 To set all options to their default values, click **Use Defaults**.
- 4 Click **OK** to close the dialog box and begin recording.

Debug Options

The Debug options let you determine the level of debug information generated during recording. The following options are available:

General Options

Enable Generic Debug Options: Enables the generic debugging options: *Class Dumping*, *Hooking Debug Level*, *Stack Trace*, and *Trace Support*. When you enable this option, VuGen performs a stack trace, even if the first **Stack Trace** option is disabled. Use the *Stack Trace* in conjunction with *Class*

Dumping to determine the context for the hooked parts in the application. The trace can help you determine where to place additional hooks. (disabled by default).

Stack Trace: Logs all invocations in a stack trace. This setting provides a Java stack trace for every recorded function. Use this option in conjunction with Class Dumping, to determine the context for the hooked parts in the application. This trace can help you solve cases where a parameter is not correlated and to determine where to place additional hooks. Note that enabling this option slows down the application (disabled by default).

Stack Trace Limit: The maximum number of calls stored in the stack. When a stack trace is enabled, and the number of calls exceeds the specified value, the stack trace is truncated. The default value is 20 calls.

Trace Support: Traces all major support calls and writes them to the *Tracer.log* file in the Vuser directory. (disabled by default).

Show Progress Window: Enables the progress window for Mercury products. (enabled by default).

Debug Class Loaders: Give debug printouts for non-system ClassLoader support. (disabled by default).

Synchronize Threads: For multi-threaded applications, instructs VuGen to synchronize between the different threads. (disabled by default).

Digest Calculation: Generate a digest of all recorded objects. (disabled by default).

Exclude from Digest: A list of objects not to be included in the digest calculation.

Debug Variables: Casts <undefined> variables to their types. Also, each variable in the variables section which is also an interface, will have a comment indicating the original type. (disabled by default).

Specify Hook: Inserts a string before the invocation of the script indicating the hook that caused it. This is useful for capturing redundant recordings. (disabled by default)

Specify Thread: Inserts a string before the invocation of the script indicating the thread that it runs in. This is useful for identifying multi-threaded application. (disabled by default)

Hooking Options

Hooking Debug Level: The level of hooking-debug printouts from within the recorder. Level 0 indicates that no debug printout will be issued.

Ignore Classes: A list of the classes to ignore. All classes containing the specified strings will be excluded from the hooking mechanism.

Printout Redirections: Determines where to redirect the printouts from the hooking mechanism. The options are the Console, a separate file, or the Debug file. The default is the Console.

Make Methods Public: Make hooked methods public. (disabled by default)

Make Class Public: Make hooked class public. (enabled by default)

Log Class Hooking: Creates a log file containing a string representation of all classes before and after hooking. This option should only be used for intense debugging, as it significantly decreases performance. (disabled by default)

Log Specific Class Hooking: A list of the classes for which a hooking log file should be generated. If no class is specified, all classes are logged.

Class Dumping Options

Class Dumping: Dumps the loaded classes to the Vuser directory. (disabled by default)

Class to Dump: A list of the classes to be dumped after hooking. Any class containing one of the specified strings will be dumped. If no class is specified, all classes are dumped.

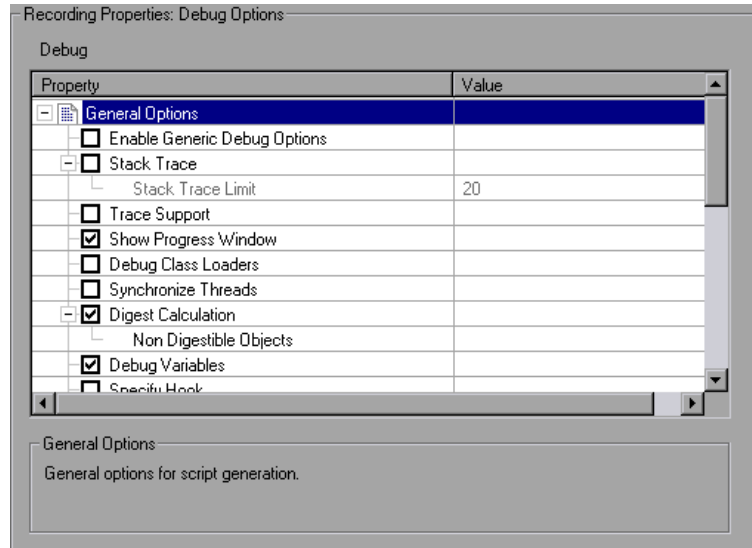
Dump Suffix: The suffix to append to the dumped class names. The default suffix is `_DUMP`.

Class Dump Directory: The directory to which to dump the classes.

Flat Class Dumping: Dump all classes into a single directory and precede each class with its full package. If this option is disabled, a directory hierarchy is created. (disabled by default)

To set the debug options:

- 1 Click **Options** in the Start Recording dialog box and select the **Recording Properties:Debug Options** node.



- 2 Enable the desired options, or for options that require values, enter the desired value.
- 3 To set all options to their default values, click **Use Defaults**.
- 4 Click **OK** to close the dialog box and begin recording.

CORBA Options

The following options are specific to the Corba-Java protocol. These options let you set the Corba specific recording properties and several callback options. The following options are available:

Record Properties: Instructs VuGen to record system and custom properties related to the protocol. By default, this option is enabled.

Show IDL Constructs: Displays the IDL construct that is used when passed as a parameter to a CORBA invocation. By default, this option is enabled.

Record DLL only: Instructs VuGen to record only on a DLL level. By default, this option is disabled.

Resolve CORBA Objects: When correlation fails to resolve a CORBA object, recreate it using its binary data. By default, this option is disabled.

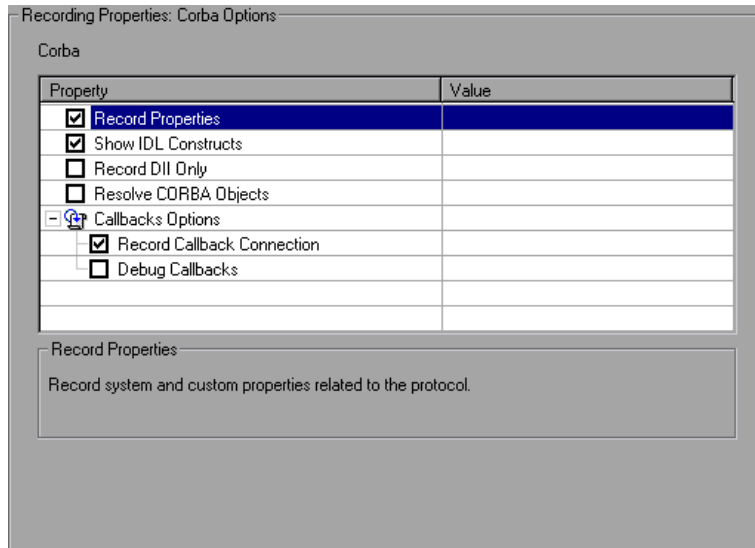
Callback Options

Record Callback Connection: Instructs VuGen to generate a connect statement for the connection to the ORB, for each callback object. By default, this option is disabled.

Debug CallBacks: Allows debugging information to be generated on callbacks. By default, this option is disabled.

To set the Corba options:

- 1 Click **Options** in the Start Recording dialog box and select the **Recording Properties:Corba Options** node.



- 2 Enable or disable the options as desired.
- 3 To set all options to their default values, click **Use Defaults**.
- 4 Click **OK** to close the dialog box and begin recording.

Correlating Java Scripts

VuGen's correlation allows you to link Java Vuser functions by using the results of one statement as input to another.

This chapter describes:

- ▶ Standard Correlation
- ▶ Advanced Correlation
- ▶ String Correlation
- ▶ Using the Serialization Mechanism

The following information only applies to Corba-Java and RMI-Java Vuser scripts.

About Correlating Java Scripts

Vuser scripts containing Java code often contain dynamic data. When you record a Corba or RMI 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 Corba 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 194.

Standard Correlation

Standard correlation refers to the automatic correlation performed during recording for simple objects, excluding object arrays, vectors, and container constructs.

When the recorded application invokes a method that returns an object, VuGen’s correlation mechanism records these objects. When you run the script, VuGen compares the generated objects to the recorded objects. If the objects match, the same object is used. The following example shows two Corba objects *my_bank* and *my_account*. The first object, *my_bank*, is invoked; the second object, *my_account*, is correlated and passed as a parameter in final line of the segment:

```
public class Actions {  
  
    // Public function: init  
    public int init() throws Throwable {  
  
        Bank my_bank = bankHelper.bind("bank", "shunra");  
        Account my_account = accountHelper.bind("account", "shunra");  
  
        my_bank.remove_account(my_account);  
    }  
:  
}
```

Advanced Correlation

Advanced or *deep* correlation refers to the automatic correlation performed during recording for complex objects, such as object arrays and Corba container constructs.

The deep correlation mechanism handles Corba constructs (structures, unions, sequences, arrays, holders, 'any's) as containers. This allows it to reference inner members of containers, additional objects, or different containers. Whenever an object is invoked or passed as a parameter, it is also compared against the inner members of the containers.

In the following example, VuGen performs deep correlation by referencing an element of an array. The *remove_account* object receives an account object as a parameter. During recording, the correlation mechanism searches the returned array *my_accounts* and determines that its sixth element should be passed as a parameter.

```
public class Actions {  
  
    // Public function: init  
    public int init() throws Throwable {  
  
        my_banks[] = bankHelper.bind("banks", "shunra");  
        my_accounts[] = accountHelper.bind("accounts", "shunra");  
  
        my_banks[2].remove_account(my_accounts[6]);  
    }  
    :  
}
```

The following segment further illustrates enhanced correlation. The script invokes the *send_letter* object that received an *address* type argument. The

correlation mechanism retrieves the inner member, *address*, in the sixth element of the *my_accounts* array.

```
public class Actions {  
  
    // Public function: init  
    public int init() throws Throwable {  
  
        my_banks = bankHelper.bind("bank", "shunra");  
        my_accounts = accountHelper.bind("account", "shunra");  
  
        my_banks[2].send_letter(my_accounts[6].address);  
    }  
:  
}
```

String Correlation

String correlation refers to the representation of a recorded value as an actual string or a variable. When you disable string correlation (the default setting), the actual recorded value of the string is indicated explicitly within the script. When you enable string correlation, it creates a variable for each string, allowing you to use it at a later point in the script.

In the following segment, string correlation is enabled—you store the value returned from the `get_id` method in a *string* type variable for use later on in the script.

```
public class Actions {

    // Public function: init
    public int init() throws Throwable {

        my_bank = bankHelper.bind("bank", "shunra");
        my_account1 = accountHelper.bind("account1", "shunra");
        my_account2 = accountHelper.bind("account2", "shunra");

        string = my_account1.get_id();
        string2 = my_account2.get_id();
        my_bank.transfer_money(string, string2);
    }
}
:
```

You set the correlation method from the **Correlation** tab in the recording options.

Correlate Strings: Correlate strings in script during recording. If you disable this option, the actual recorded values are included in the script between quotation marks. If this option is disabled, all other correlation options are ignored. (disabled by default)

Correlate String Arrays: Correlate strings within string arrays during recording. If you disable this option, strings within arrays are not correlated and the actual values are placed in the script. (enabled by default)

Advanced Correlation: Enables correlation on complex objects such as arrays and Corba container constructs and arrays. This type of correlation is also known as *deep* correlation. (enabled by default)

Correlation Level: Determines the level of deep correlation—how many inner containers to search.

Correlate Collection Type: Correlate objects contained in a Collection class for JDK 1.2 or higher. (disabled by default)

Using the Serialization Mechanism

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

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

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user directory. Additional parameters are saved as new binary data files, numbered sequentially. VuGen generates the following code:

```
public class Actions {

    // Public function: init
    public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
        map.set_point(p);
    }
:
}
```

The integer passed to `lr.deserialize` is 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 13, "Setting Java Recording Options")

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 number of binary file to load.

If you choose not to expand objects in your script by clearing the **Unfold Serialized Objects** check box, then 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:

<i>true</i>	Use VuGen's serialization mechanism.
<i>false</i>	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 *Actions* 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)Ir.deserialize(_string,0);
        map.set_vector(_vector);
    }

    :
}
```

Due to the complexity of the serialization mechanism, which opens up the objects to ASCII representation, opening large objects while recording may increase the time required for script generation. To decrease this time, you can specify flags which will improve the performance of the serialization mechanism.

When adding **Ir.deserialize** to your script, it is recommended 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 would 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

For complete information on the recording options, see Chapter 13, “Setting Java Recording Options.”

15

Configuring Java Run-Time Settings

After you record a Java Vuser script, you configure the run-time settings for the Java Virtual Machine.

This chapter describes:

- ▶ Understanding the Java VM Settings
- ▶ Classpath Options
- ▶ Specifying the Virtual Machine Settings

The following information applies to Java, EJB Testing, Corba-Java, and RMI-Java type Vusers.

About Java Run-Time Settings

After developing a Java Vuser script, you set the run-time settings for the Java VM (Virtual Machine). These settings let you set additional paths and parameters, and determine the run mode.

You set the Java related run-time settings through the **Java VM** options in the Run-Time Settings dialog box.



To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar. You can also modify the run-time settings from the ProTune Console. In the Console window, select the script whose setting you want to modify, and click the **Run-Time settings** button.

This chapter only discusses the Run-Time settings for Java type Vusers—Java, EJB Testing, Corba-Java, and RMI-Java. For information about run-time settings that apply to all Vusers, see Chapter 9, “Configuring Run-Time Settings.”

Understanding the Java VM Settings

In the Java VM section, you provide information about the Java virtual machine settings. The following settings are available:

Additional VM Parameters: Enter any optional parameters used by the virtual machine.

When you run a Vuser, VuGen automatically sets the *Xbootclasspath* variable. You use this dialog box to specify parameters, in addition to the ones defined in *Xbootclasspath*. If you specified additional VM parameters for recording, you can instruct VuGen to save the parameters and use them during replay.

Classpath Options

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper replay.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.

To set the Classpath run-time settings:

- 1 Open the Run-Time settings (F4). Select the **Java Environment Settings:Classpath** node in the Run-Time settings tree.

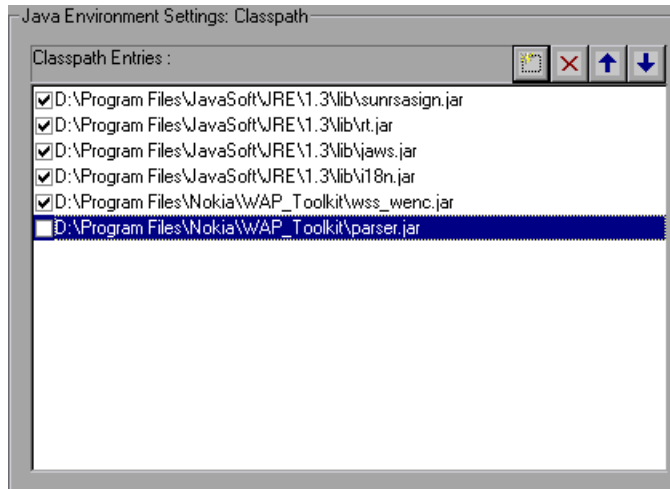





- 2 To add a classpath to the list:

Click the Add Classpath button. VuGen adds a new line to the classpath list.

Type in the path and name of the *jar*, *zip* or other archive file for your class. Alternatively, click the **Browse** button to the right of the field, and locate

the desired file. VuGen adds the new location to the classpath list, with an enabled status.

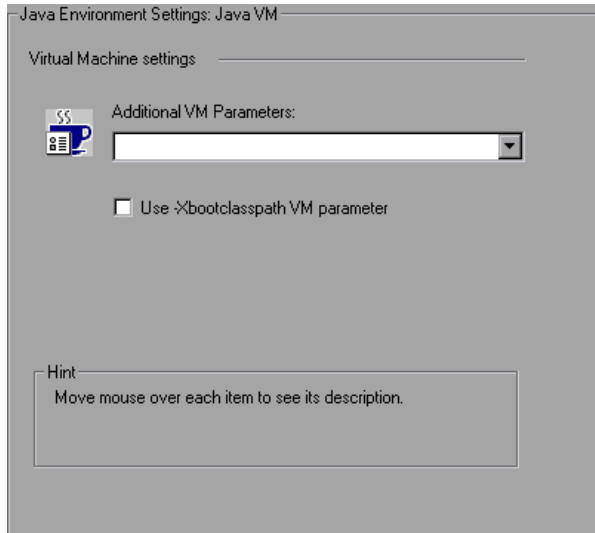


-  **3** To permanently remove a classpath entry, select it and click the Delete button.
- 4** To disable a classpath entry for a specific test, clear the check box to the left of the entry.
-  **5** To move a classpath entry down in the list, select it and click the Down arrow.
-  **6** To move a classpath entry up within the list, select it and click the Up arrow.
- 7** Click **OK** to close the dialog box.

Specifying the Virtual Machine Settings

To set the Java VM run-time settings:

- 1 Choose **Vuser > Run-Time Settings** and select the **Java Environment Settings:Java VM** node in the Run-Time Settings tree.



- 2 In the **Additional VM Parameters** box, enter any optional parameters used by the Load Generator machine.
- 3 To replay with the *-Xbootclasspath/p* option, select the **Use -Xbootclasspath VM parameter** option.
- 4 Click **OK**.

Part IV

Application Deployment Solution Protocols

16

Developing Citrix Vuser Scripts

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.

This chapter describes:

- ▶ Getting Started with Citrix Vuser Scripts
- ▶ Setting the Recording Options
- ▶ Synchronizing Replay
- ▶ Using Citrix Functions
- ▶ Viewing a Citrix Vuser Script
- ▶ Setting the Citrix Display Settings
- ▶ Setting the Citrix Run-Time Settings
- ▶ Understanding ICA Files
- ▶ Disconnecting from the Citrix Server
- ▶ Tips for Working with Citrix Vuser Scripts

The following information applies to the Citrix ICA protocol.

About Recording 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 an **ctrx** prefix. These functions emulate the analog movements of the mouse and keyboard. In addition, the **ctrx** functions allow you to synchronize the replay of the actions, by waiting for specific windows to open.

VuGen also allows you to record a Citrix NFUSE session. The NFUSE session works with a browser instead of a client. In order to record NFUSE sessions, you must perform a multi-protocol recording for Citrix and Web Vusers. (See Chapter 3, “Recording with VuGen.”) In multi-protocol mode, VuGen generates functions from both protocols during the recording session.

In the following example, **ctrx_mouse_click** simulates a mouse click on the left button.

```
ctrx_mouse_click(44, 318, LEFT_BUTTON, 0);
```

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 the window, 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, refer to “Tips for Working with Citrix Vuser Scripts,” on page 226.

To develop a Citrix ICA script:

1 Record the actions using VuGen.

Invoke VuGen and create a new Vuser script. Note that when recording a Citrix ICA client, you may work with a single protocol Vuser script, Citrix ICA. When recording an NFUSE session, you must create a multi-protocol Vuser script Citrix ICA and HTTP, which enables the recording of both protocols.

When you record a Citrix ICA client session, you must specify the Mercury Interactive version of the client as the application to record in the **Program to record** box. This file is called *runDlg.exe* and it is located in the *bin* directory of the ProTune installation.

For more details about recording, see Chapter 3, “Recording with VuGen.”

1 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

2 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

3 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 220.

4 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include the pacing, logging, think time, and connection information.

For information about the general run-time settings, refer to Chapter 9, “Configuring Run-Time Settings.” For details about the Citrix specific run-time settings, see “Setting the Citrix Run-Time Settings,” on page 221.

5 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 stand-alone test, see “Tips for Working with Citrix Vuser Scripts,” on page 226 and Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

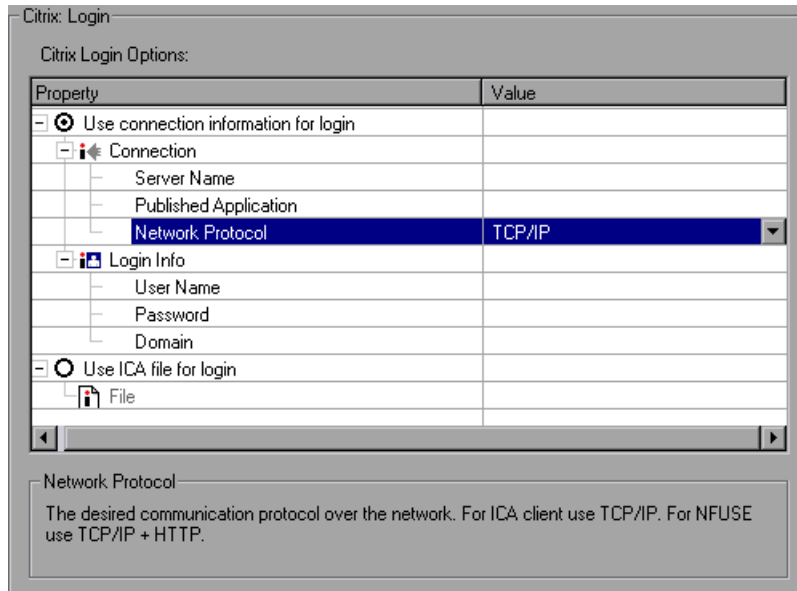
After you create a Citrix Vuser script, you integrate it into a session step. For more information, refer to the.

Setting the Recording Options

You use the **Citrix:Login** recording options to set the connection parameters used by VuGen to connect to the remote server.

To set the Citrix recording options:

- 1 Open the Recording Options dialog box. Choose **Tools > Recording Options** or click the **Options** button in the Start Recording dialog box. The keyboard shortcut key is CTRL+F7. Select the **Citrix:Login** node.



- 2 If you have an existing *.ica* file with all of the relevant configuration information, select **Use ICA file for login**. In the following row, specify the full path of the ICA file, or click the Browse button and locate the file on the

local disk or network. For information about the format of an ICA file, see “Understanding ICA Files,” on page 223. Skip to step 6.

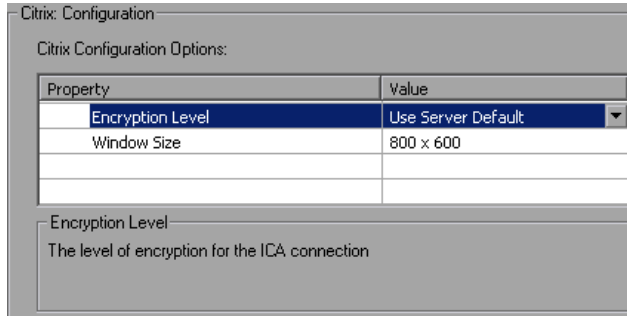
- 3 If you do not have an ICA file, select **Use connection info for login**. This is the default setting.
- 4 In the **Connection** section, enter the connection information:
 - the Citrix server name
 - The name of the published application as it is recognized on Citrix server. If you do not specify a published application, VuGen uses the server’s desktop.
 - Select a **Network Protocol**: TCP/IP or TCP/IP+HTTP.
- 5 In the **Login Info** section, enter the login information:
 - the login name for the Citrix server
 - the password for the Citrix server. This field is encrypted.
 - the domain of the Citrix server

You must provide the name of the server—otherwise the connection will fail. If you provide login information, it is recorded into the `ctx_connect_server` function:

```
ctx_connect_server("steel", "test", "test", "testlab");
```

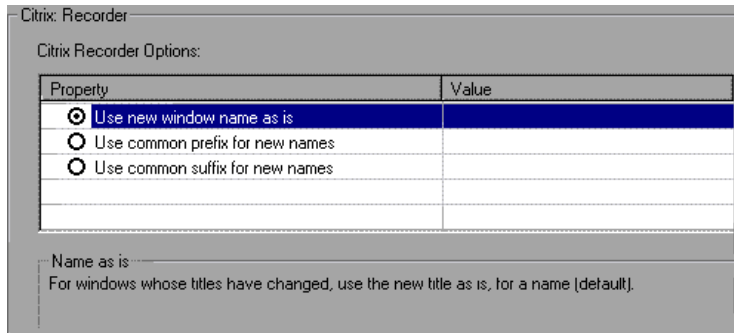
If you do not provide login information, you are prompted for the information when the client locates the specified server.

6 Select the **Citrix:Configuration** node, and set the desired options:



- **Encryption Level:** The level of encryption for the ICA connection: **Basic**, **128 bit for login only**, **40 bit**, **56 bit**, **128 bit**, or **Use Server Default** to use the server's default.
- **Window Size:** The size of the client window: **640 x 480**, **800 x 600** (default), or **1024 x 768**.

7 Select the **Citrix:Recorder** node. Specify how to generate window names for windows whose titles change during the recording session.



- Select **Use new window name as is**, to use the current window title as is, for a name. (default)
- Select **Use common prefix for new names**, to use the common string from the beginning of the titles (the prefix) as a name.
- Select **Use common suffix for new names**, to use the common string from the end of the titles (the suffix) as a name.

- 8 When recording an NFUSE session, set the Web recording mode to URL-based. Choose the **Internet Protocol:Recording** node in the recording options. Select **URL-based script**.
- 9 Click **OK** to accept the setting and close the dialog box.

Synchronizing Replay

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 functions that help synchronize the Vuser's replay of the script: **ctx_sync_on_window**.

The **ctx_sync_on_window** function 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). For non-window objects, such as menus that never become fully activated, VuGen usually generates a function with the *Create* event. For standard windows, VuGen generates a function with the *Active* event.

For all windows recorded with the **ctx_sync_on_window** function, VuGen saves a screen shot and stores it in the script's `data/bitmap` directory. The function's seventh argument is the name of the bitmap file.

```
ctx_sync_on_window("ICA Administrator Toolbar", ACTIVATE, 768, 0, 33,
  573, "129c54d64c9679ea1a0cb8bb02c2a981", CTRX_LAST);
```

Manual Synchronization

You can also add manual synchronization through VuGen's user interface or by inserting custom synchronization functions.

The first type of manual synchronization is through VuGen's user interface. During recording, VuGen provides a marker tool on the recording floating toolbar. Using this tool, you mark a bitmap area in the client window that

needs to be in focus before resuming replay. A common use of this capability, is a specific graphic image in a browser window. During the recording session, you mark an area as a bitmap using VuGen’s marker tool, located on the floating toolbar. VuGen generates a **ctrx_sync_on_bitmap** function.

```
ctrx_sync_on_bitmap(93, 227, 78, 52,
    "66de3122a58baade89e63698d1c0d5dfa");
```



During replay, it looks for the bitmap at the specified location. To mark a section for synchronization, click the marker button. Mark the section of the window for which you want to wait, by dragging the mouse from the top left of the section to the bottom right. VuGen generates a **ctrx_sync_on_bitmap** function with the selected coordinates as arguments.

To implement additional types of synchronization, you must manually enter one of the following functions into your script.

- **ctrx_sync_on_bitmap_change**: You specify an area (location and size), in which VuGen waits for a change. This is useful when you do not know what data or image will be displayed in the area, but you do expect it to change. To assist you in deriving the correct coordinates for the bitmap area, you can record a **ctrx_sync_on_bitmap** function using the marker tool (see above), and manually modify the function name and remove the fifth argument.

The syntax of the functions are as follows:

```
ctrx_sync_on_bitmap (x_start, y_start, width, height, hash);
ctrx_sync_on_bitmap_change (x_start, y_start, width, height,
    [initial_wait_time,] [timeout,] CTRX_LAST);
```

You can add optional arguments to **ctrx_sync_on_bitmap_change**: initial wait time value—when to begin checking for a change, and a timeout—the amount of time in seconds to wait for a change to occur before failing.

```
/* recorded function */
ctrx_sync_on_bitmap(93, 227, 78, 52,
    "66de3122a58baade89e63698d1c0d5dfa");
```

```
/* modified function with an initial wait time of 300 and timeout of 400*/
ctx_sync_on_bitmap_change(93, 227, 78, 52, 300, 400, CTRX_LAST);
```

During recording, the bitmaps generated for the `ctx_sync_on_bitmap` function are saved under the script's `data/bitmap` directory. The bitmap name is 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 tuning session, wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.

- **ctx_set_waiting_time:** 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 `ctx_set_window` functions are timing out, you can increase the default timeout which is 60 seconds, to 180 seconds.

```
ctx_set_waiting_time(180);
```

- **ctx_win_exist:** Checks if a window is visible in the Citrix client. For example, you could use this function to check if a browser was launched. The second argument indicates how long the function should wait for the window to become visible. If it was not launched, you perform a double-click on its icon.

```
if (!ctx_win_exist("Welcome to MSN.com- Microsoft Internet
Explorer",6))
    ctx_mouse_double_click(34, 325, LEFT_BUTTON, 0)
```

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

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 more information about the **ctx** functions, refer to 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, but you may only use one wildcard per window name.

Citrix Functions

ctx_connect_server	Connects to a remote server using the Citrix client.
ctx_disconnect_server	Closes the connection to a server.
ctx_get_bitmap_value	Gets the hash value of the specified bitmap.
ctx_get_window_name	Gets the name of the window in focus.
ctx_get_window_position	Gets the position of the specified window, or of the window in focus.
ctx_key	Emulates the pressing of a non-alphanumerical key.
ctx_key_down	Emulates the pressing of a key on the keyboard.
ctx_key_up	Emulates the releasing of a keyboard key.
ctx_mouse_click	Emulates a mouse click.
ctx_mouse_double_click	Emulates a mouse double-click.
ctx_mouse_down	Emulates the pressing of a mouse button.

ctrx_mouse_move	Emulates a mouse movement.
ctrx_mouse_up	Emulates the release of a mouse button.
ctrx_nfuse_connect	Connects to a Citrix server via an NFUSE portal.
ctrx_set_connect_opt	Sets the connection options.
ctrx_set_exception	Specifies exception handling.
ctrx_set_waiting_time	Sets the waiting time for all subsequent timing functions
ctrx_set_window	Waits for the specified window to open.
ctrx_set_window_ex	Waits a specific number of seconds for the specified window to open.
ctrx_sync_on_bitmap	Waits until the bitmap specified by the coordinates, is displayed.
ctrx_sync_on_bitmap_change	Waits until the area specified by the coordinates, changes.
ctrx_sync_on_window	Waits for a window to be created or active.
ctrx_type	Emulates the typing of alphanumerical keys.
ctrx_unset_window	Waits for the specified window to close.
ctrx_wait_for_event	Waits for a specified event to occur.
ctrx_win_exist	Checks whether the specified window exists.

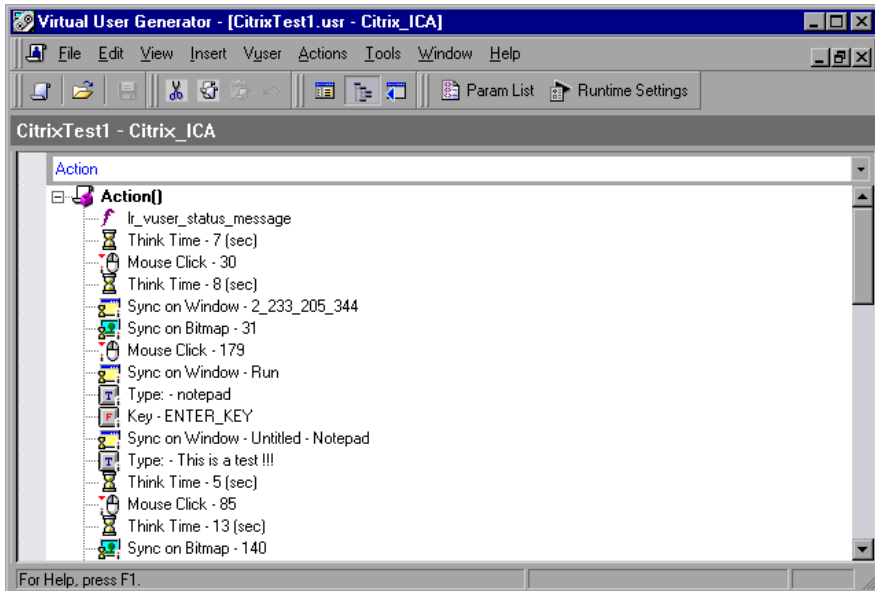
Viewing a Citrix Vuser Script

You can view the contents of the *vuser_init*, *Actions*, and *vuser_end* sections in the VuGen script editor. To display an action, select the action name in the left pane. VuGen displays its contents in the right pane. When viewing and editing a Citrix Vuser script in VuGen, you choose between viewing the script in the icon-based tree view or the text-based script view.

To display the tree view:



From the VuGen main menu, select **View > Tree View**, or click the **View script as tree** icon. The Actions section of the Vuser script is displayed in the icon-based tree view. If you are already in the tree view, the menu item is disabled.



To display the script view:



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

```

Virtual User Generator - [CitrixTest1.usr - Citrix_ICA]
File Edit View Insert Vuser Actions Tools Window Help
Param List Runtime Settings
CitrixTest1 - Citrix_ICA
vuser_init
Action
vuser_end
globals.h

ctrx_mouse_click(30, 12, LEFT_BUTTON, 0, \"-2_572_805_31\");
lr_think_time(8);
ctrx_sync_on_window(\"2_233_205_344\", CREATE, 2, 233, 205, 344, \"aef84...);
ctrx_sync_on_bitmap(31, 402, 233, 35, \"dcd1bb99773d579c1a4a434a4e95F9c...);
ctrx_mouse_click(179, 508, LEFT_BUTTON, 0, NULL);
ctrx_sync_on_window(\"Run\", ACTIVATE, 5, 389, 348, 180, \"aead7e4e4a220f...);
ctrx_type(\"notepad\");
ctrx_key(\"ENTER_KEY\", 0);
ctrx_sync_on_window(\"Untitled - Notepad\", ACTIVATE, 63, 128, 601, 433, ...);
ctrx_type(\"This is a test !!!\");
  
```

To learn more details about the recording session, you can view a log of the actions that were generated during recording. Choose **View > Output Window** and select the **Recording Log** tab. VuGen displays a detailed log of all actions performed by VuGen.

```

Execution Log Recording Log Recorded Event Log
[Citrix Recorder ( 8fc: 848)] ctrx_key_down(13, 0);
[Citrix Recorder ( 8fc: 848)] ctrx_key_up(13, 0);
[Citrix Recorder ( 8fc: 848)] ctrx_mouse_down(1, 0, 118, 30, Untitled - Notepad);
[Citrix Recorder ( 8fc: 848)] ctrx_mouse_up(1, 0, 118, 30, Untitled - Notepad);
[Citrix Recorder ( 8fc: 848)] ctrx_create_window name=171_169_118_50 flag=3 style=-;
[Citrix Recorder ( 8fc: 848)] ctrx_sync_on_bitmap(182, 198, 91, 43, \"b697f65ce7d2e0...);
[Citrix Recorder ( 8fc: 848)] ctrx_sync_on_window(\"171_169_118_50\", 171, 169, 118, ...);
[Citrix Recorder ( 8fc: 848)] ctrx_mouse_down(1, 0, 172, 79, Untitled - Notepad);
[Citrix Recorder ( 8fc: 848)] ctrx_mouse_up(1, 0, 172, 79, Untitled - Notepad);
[Citrix Recorder ( 8fc: 848)] ctrx_create_window name=About Notepad flag=0 style=-1...;
[Citrix Recorder ( 8fc: 848)] ctrx_active_window(\"About Notepad\");
  
```

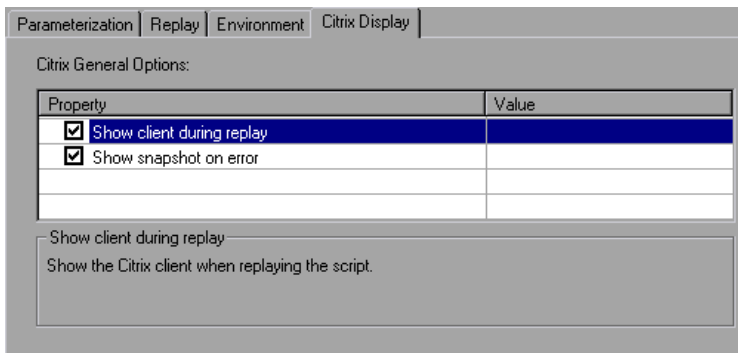
Setting the Citrix Display Settings

Before running your Citrix Vuser script, you can set several display options that will be used during replay. Although these options increase the load upon the server, they are useful for debugging and analyzing your session.

You set the Citrix display settings from the General Options dialog box.

To set the Citrix display options:

- 1** Open the General Options dialog box. Choose **Tools > General Options**.
- 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 snapshot on error** to display a snapshot of the screen when an error occurs.
- 5** Click **OK**.

Setting the Citrix Run-Time Settings

After creating a Citrix Vuser script, you set the run-time settings. These settings let you control the behavior of the Vuser when running the script. Your Citrix run-time settings in the **Configuration** node, should correspond to the properties of your Citrix client. These settings will influence the load upon the server. To view the connection properties, select the icon representing the ICA connection in the Citrix Program Neighborhood, and choose **Properties** from the right-click menu. Select the **Default Options** tab.

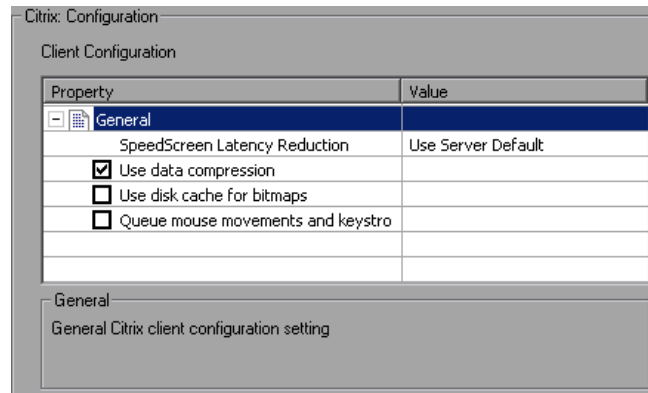
Note: Citrix Vusers do not support modem speed emulation or IP spoofing.

You set the Citrix run-time settings from the Run-Time Settings dialog box.

To set the Citrix Run-Time Settings:



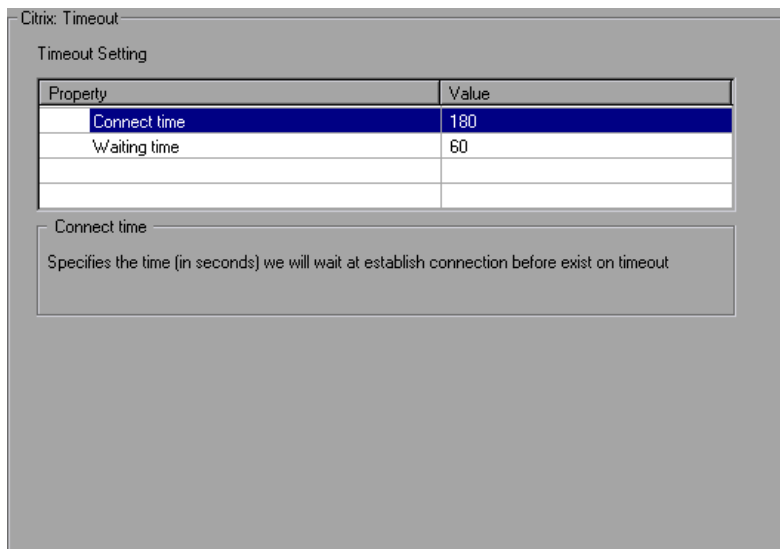
- 1** Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the VuGen toolbar, or choose **Vuser > Run-Time Settings**.
- 2** Select the **Citrix:Configuration** node. Specify the **General** properties:



- **SpeedScreen Latency Reduction:** The mechanism used to enhance user interaction when the network speed is slow. You can turn this mechanism **on** or **off**, depending on the network speed. The **auto** option turns it on or off based on the current network speed. If you do not know

the network speed, set this option to **Use Server Default** to use the machine's default.

- 3 Set the **Use data compression** option. This option instructs the Vuser to compress the transferred data. To enable this option, select the check box to the left of the option; to disable it, clear the check box. You should enable data compression if you have a limited bandwidth. (enabled by default)
- 4 Set the **Use disk cache for bitmaps** option. This option instructs the Vuser to use a local cache to store bitmaps and commonly-used graphical objects. To enable this option, select the check box to the left of the option; to disable it, clear the check box. You should enable this option if you have a limited bandwidth. (disabled by default)
- 5 Set the **Queue mouse movements and keystrokes** option. This option instructs the Vuser to create a queue of mouse movements and keystrokes, and send them as packets to the server less frequently. The purpose of this is to reduce the network traffic with slow connections. Enabling this option makes the session less responsive to keyboard and mouse movements. To enable this option, select the check box to the left of the option; to disable it, clear the check box. (disabled by default)
- 6 Set the timeout options. Select the **Citrix:Timeout** node.



- **Connect Time:** The time in seconds to wait idly at an established connection before exiting. The default is 180 seconds.
 - **Waiting Time:** The time in seconds to wait idly at a synchronization point before exiting. The default is 60 seconds.
- 7 Click **OK** to accept the setting and close the dialog box.

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 injector 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
```

Disconnecting from the Citrix Server

After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time the same client opens a new connection. Consequently, a new connection by the same client will face the same workspace it disconnected from previously. This is not the

preferred way to run tests where, typically, each new run of a test requires a clean workspace.

The solution is to configure the Citrix server not to save the previous session but to completely disconnect from the client each time the client breaks the connection or times-out.

To force Citrix Server disconnection for the XP server:

- 1 Open the Citrix Connection Configuration dialog box. Choose **Start > Programs > Citrix > MetaFrame XP > Citrix Connection Configuration**.
- 2 Double-click on the ica-tcp connection name. The Edit Connection dialog box appears.
- 3 Click on the **Advanced** button. The Advanced Connection Settings dialog box appears.

Advanced Connection Settings

Logon
 Disabled Enabled

Timeout settings (in minutes)
 Connection: No Timeout
 (inherit user config)
 Disconnection: No Timeout
 (inherit user config)
 Idle: No Timeout
 (inherit user config)

Security
 Required encryption:
 Use default NT Authentication

AutoLogon
 User Name:
 Domain:
 Password:
 Confirm Password:
 Prompt for Password:
 (inherit client config)

Initial Program
 Command Line:
 Working Directory:
 (inherit client/user config)
 Only run Published Applications

User Profile Overrides
 Disable Wallpaper

On a broken or timed-out connection, the session. (inherit user config)
 Reconnect sessions disconnected (inherit user config)
 Shadowing (inherit user config)

OK Cancel Help

- 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*.

Tips for Working with Citrix Vuser Scripts

Recording Tips

- 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.
- It is recommended that you do not move or resize windows during the recording session.
- Vugen uses the Desktop's color settings.
- In order 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 Injector machines, check the settings of the ICA client, and make sure that they are consistent between all injector and recording machines.
- Supported resolutions (window sizes) are 640 x 480, 800 x 600, and 1024 x 768. Display settings of 1024 x 768 are recommended on the recording machine as it will allow the Citrix window, whose default size is 800 x 600, to be displayed properly.
- While waiting for an event during recording, such as an HTML page to load, it is recommended that you add manual synchronization points with the **ctx_sync_on_bitmap** function. For details, see "Synchronizing Replay" on page 213.
- Configure the Citrix server to completely close a session. See "Disconnecting from the Citrix Server," on page 224.
- Record the connection process into the *vuser_init* section, and the closing process in the *vuser_end* section. This will prevent you from performing iterations on the connection process.

- If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine—the Mercury Interactive version of the client. 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.
- When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, **Start > Programs > Microsoft Word**, be sure to click on the word *Programs*.
- Disable client updates when prompted by the Citrix client.

Replay Tips

- 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.
- For best results, do not disable think time in the run-time settings. Think time is especially relevant before **ctrx_sync_on_window** and **ctrx_sync_on_bitmap** functions, which require time to stabilize.
- 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 choose **Application Set Settings** or **Custom Connection Settings** when selecting an ICA connection icon. from the right-click menu. Click on the Default Options tab.
- Machines running Citrix 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 a Terminal Server session on the machine. You relate to this Terminal Server as a new injector machine. To indicate this virtual injector machine from ProTune, use the following format: *machine_name:1, machine_name:2, ...* using either the machine name or its IP address. Note that sessions on a Terminal server use, by default, 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

- Add breakpoints to your script in VuGen to help you determine the problematic lines of code.
- 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 add a delay manually using **lr_think_time**, it is recommended that you use one of the synchronization functions discussed in “Synchronizing Replay” on page 213.
- You can view additional replay information in the Extended log. You enable Extended logging from the Run-Time settings (F4 Shortcut key) **Log** tab. You can view this information in the Execution Log tab, or the *output.txt* file in the script’s directory.
- When an error occurs, VuGen saves a snapshot of the screen to the script’s *output* directory. You can view the bitmap to try to determine why the error occurred.
- During recording, the bitmaps generated for the **ctrx_sync_on_bitmap** function, are saved under the script’s *data* directory. The bitmap name has the format of *hash_value.bmp*. If synchronization fails during replay, the generated bitmap is written to the script’s output directory, or if you are running it in a tuning session, wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.
- To show all Vusers during a tuning session run, enter the following in the Vuser command line box: `-lr_citrix_vuser_view`. In ProTune, 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 check the version of the server, make sure the MetaFrame XP or 1.8 server is installed. Select Citrix Connection Configuration on the server’s console toolbar and choose **Help > About**.

Part V

Client Server Protocols

17

Developing Database Vuser Scripts

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 describes:

- ▶ Introducing Database Vusers
- ▶ Understanding Database Vuser Technology
- ▶ Getting Started with Database Vuser Scripts
- ▶ Setting Database Recording Options
- ▶ Using LRD Functions
- ▶ Understanding Database Vuser Scripts
- ▶ Evaluating Error Codes
- ▶ Handling Errors

The following information applies to Client Server Database (CtLib, DbLib, Informix, MS SQL Server, Oracle, and ODBC, DB2-CLI) and ERP Siebel Vuser scripts only.

About Recording Database Vuser Scripts

When you record a database application communicating with a server, VuGen generates a Database Vuser script. VuGen supports the following database types: CtLib, DbLib, Informix, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity. Each LRD function has an **lrd** prefix and represents one or more database functions. For example, the **lrd_fetch** function represents a fetch operation.

When you run a recorded session, the Vuser script communicates directly with the database server, performing the same operations as the original user. You can set the Vuser behavior (run-time settings) to indicate the number of times to repeat the operation and the interval between the repetitions. For more information, see Chapter 9, “Configuring Run-Time Settings.”

Using VuGen, you can parameterize a script, replacing recorded constants with parameters. For more information, see Chapter 7, “Defining Parameters.”

In addition, you can correlate queries or other database statements in a script, linking the results of one query with another. For more information, see Chapter 8, “Correlating Statements.”

For troubleshooting information and scripting tips, see Chapter 64, “VuGen 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:

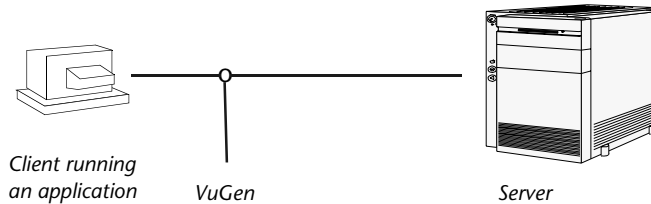
- connect to the server
- submit an SQL query
- retrieve and process the information
- disconnect from the server

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.

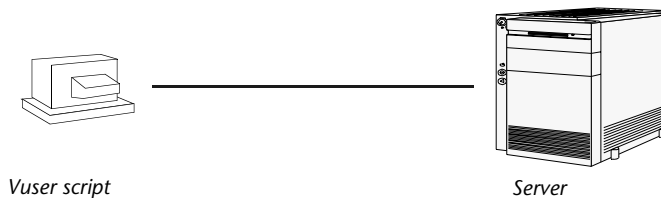
The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. Using the Console, you assign the script to multiple Vusers. The Vusers execute the script and emulate user load on the client/server system. ProTune generates performance data which you can analyze in report and graph format.

Understanding Database Vuser Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.



Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.



You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and UNIX environments. For information about recording scripts, see Chapter 3, “Recording with 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). Choose an application to record and set the recording options. Record typical operations on your application.

For details, see Chapter 3, “Recording with VuGen.”

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same query many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Correlate queries (optional).

Correlating database statements allows you to use the result of a query in a subsequent one. This feature is useful when working on a database with user constraints.

For details, see Chapter 8, “Correlating Statements.”

5 Configure the run-time settings.

The run-time settings control the Vuser script behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Database Vuser script, you integrate it into a session step on either a Windows or UNIX platform. For more information on integrating Vuser scripts in a session step, refer to your *Online Function Reference*.

Setting Database Recording Options

Before you record a database session, you set the recording options. You can set basic recording options for automatic function generation, script options, and think time:

Automatic Transactions: You can instruct VuGen to mark every **lrd_exec** and **lrd_fetch** function as a transaction. When these options are enabled, VuGen inserts **lr_start_transaction** and **lr_end_transaction** around every **lrd_exec** or **lrd_fetch** function. By default, automatic transactions are disabled.

Script Options: You can instruct VuGen to generate comments into recorded scripts, describing the **lrd_stmt** option values. In addition, you can specify the maximum length of a line in the script. The default length is 80 characters.

Think Time: VuGen automatically records the operator’s think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an **lr_think_time** statement before LRD functions. If the recorded think time is below the threshold level, an **lr_think_time** statement is not generated. The default value is five seconds.

In addition to the basic recording options, you can configure advanced options for the log file detail, CtLib specific functions, buffer size, and the recording engine:

Recording Log Options: You can set the detail level for the trace and ASCII log files. The available levels for the trace file are *Off*, *Error Trace*, *Brief Trace*, or *Full Trace*. The error trace only logs error messages. The Brief Trace logs errors and lists the functions generated during recording. The Full Trace logs all messages, notifications, and warnings.

You can also instruct VuGen to generate ASCII type logs of the recording session. The available levels are *Off*, *Brief detail*, and *Full detail*. The Brief detail logs all of the functions, and the Full detail logs all of the generated functions and messages in ASCII code.

CtLib Function Options: You can instruct VuGen to generate a send data time stamp or an extended result set statement.

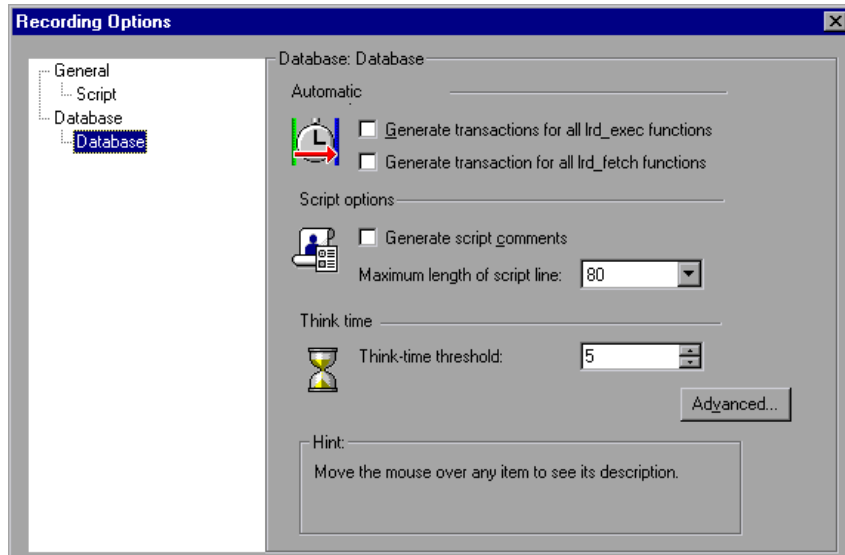
- ▶ **Time Stamp:** By default, VuGen generates `lrd_send_data` statements with the *TotalLen* and *Log* keywords for the *mpszReqSpec* parameter. The Advanced Recording Options dialog box lets you instruct VuGen to also generate the *TimeStamp* keyword. If you change this setting on an existing script, you must regenerate the Vuser script by choosing **Tools > Regenerate**. It is not recommended to generate the *Timestamp* keyword by default. The timestamp generated during recording is different than that generated during replay and script execution will fail. You should use this option only after a failed attempt in running a script, where an `lrd_result_set` following an `lrd_send_data` fails. The generated timestamp can now be correlated with a timestamp generated by an earlier `lrd_send_data`.
- ▶ **Extended Result Set:** By default, VuGen generates an `lrd_result_set` function when preparing the result set. This setting instructs VuGen to generate the extended form of the `lrd_result_set` function, `lrd_result_set_ext`. In addition to preparing a result set, this function also issues a return code and type from *ct_results*.

Code Generation Buffer Size: Specify in kilobytes the maximum size of the code generation buffer. The default value is 128 kilobytes. For long database sessions, you can specify a larger size.

Recording Engine: VuGen version 6.0 and higher utilizes a new recording engine. You can instruct VuGen to record scripts with the older recording engine for compatibility with previous versions of VuGen.

To set the Database recording options:

- 1 Choose **Tools > Recording Options**. The Recording Options dialog box opens.

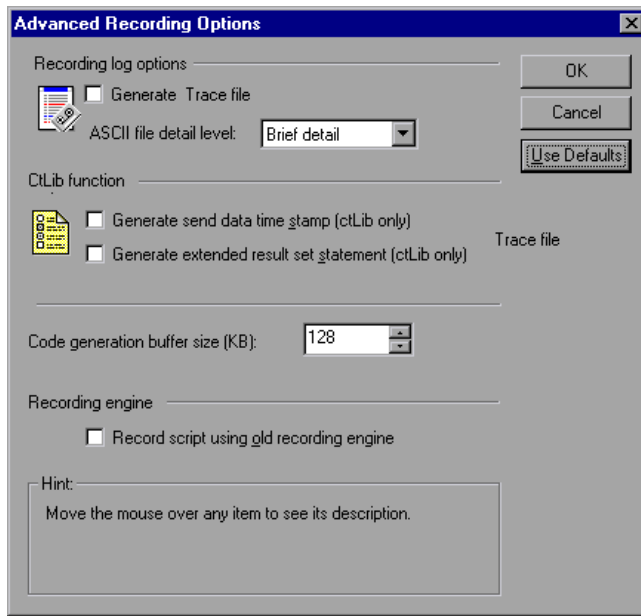


- 2 Select **Generate transactions for all lrd_exec functions** to enable automatic transactions for **lrd_exec** statements.
Select **Generate transaction for all lrd_fetch functions** to enable automatic transactions for **lrd_fetch** statements.
- 3 Select **Generate script comments** to instruct VuGen to insert descriptive comments within the script.
- 4 To change the maximum length of a line in the VuGen editor, specify the desired value in the **Maximum length of script line** box.
- 5 To change the think-time threshold value from the five second default, specify the desired value in the **Think-time threshold** box.

You can also set advanced recording options relating to the trace level, Ctlib function generation, and the code generation buffer.

To set advanced recording options:

- 1 Click the **Advanced** button in the Recording Options dialog box. The Advanced Recording Options dialog box opens.



- 2 To generate a trace file, select **Generate trace file**.
- 3 To generate an ASCII log file, select the desired detail level from the **ASCII file detail level** box.
- 4 For CtLib: To instruct VuGen to generate the TimeStamp keyword for **Ird_send_data** functions, select the **Generate send data time stamp** option.
- 5 For CtLib: To instruct VuGen to generate **Ird_result_set_ext** instead of **Ird_result_set**, select the **Generate extended result set statement** option.
- 6 To modify the size of the code generation buffer from the default value of 128 kilobytes, enter the desired value in the **Code generation buffer size** box.

- 7 To use the old VuGen recording engine to allow backwards compatibility, select the **Record script using old recording engine** option.
- 8 Click **OK** to save your settings and close the Advanced Recording Options dialog box.

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, and ODBC). You can also manually program any of the functions into your script. For syntax and examples of the LRD functions, see the *Online Function Reference* (**Help > Function Reference**).

Access Management Functions

<code>lrd_alloc_connection</code>	Allocates a connection structure.
<code>lrd_close_all_cursors</code>	Closes all open cursors.
<code>lrd_close_connection</code>	Disconnects (logs out) from the database.
<code>lrd_close_context</code>	Closes a context.
<code>lrd_close_cursor</code>	Closes a database cursor.
<code>lrd_ctlib_cursor</code>	Specifies a CtLib cursor command.
<code>lrd_commit</code>	Commits the current transaction.
<code>lrd_db_option</code>	Sets an option for the current database.
<code>lrd_free_connection</code>	Frees a connection structure.
<code>lrd_rollback</code>	Rolls back the current transaction.
<code>lrd_open_connection</code>	Connects (logs on) to the database.
<code>lrd_open_context</code>	Opens a context.
<code>lrd_open_cursor</code>	Opens a database cursor.

LRD Environment Functions

<code>lrd_msg</code>	Issues an output message.
<code>lrd_option</code>	Sets an LRD option.
<code>lrd_end</code>	Closes the lrd environment.
<code>lrd_init</code>	Initializes the lrd environment.

Retrieval Handling Functions

<code>lrd_col_data</code>	Sets a pointer indicating the location of data.
<code>lrd_fetch</code>	Fetches the next row in the result set.
<code>lrd_fetchx</code>	Fetches the next row in the result set using an extended fetch (ODBC only).
<code>lrd_result_set</code>	Returns a result set (CtLib only).

<code>lrd_result_set_ext</code>	Returns a CtLib result code and result type (extended).
<code>lrd_fetch_adv</code>	Fetches multiple rows from a result set using an extended fetch. (ODBC only)
<code>lrd_reset_rows</code>	Prepares fetched rows for an Update operation. (ODBC only)
<code>lrd_row_count</code>	Returns the number of the rows affected by an UPDATE, DELETE or INSERT statement. (ODBC, DB2)

Statement Handling Functions

<code>lrd_bind_col</code>	Binds a host variable to an output column.
<code>lrd_bind_cols</code>	Binds a host variable array to columns.
<code>lrd_bind_cursor</code>	Binds a cursor to a place holder.
<code>lrd_bind_placeholder</code>	Binds a host variable or array to a place holder.
<code>lrd_cancel</code>	Cancels the previous statement.
<code>lrd_data_info</code>	Gets I/O information. (CtLib only)
<code>lrd_dynamic</code>	Specifies a dynamic SQL statement to be processed (CtLib only).
<code>lrd_exec</code>	Executes the previously specified SQL statement.
<code>lrd_send_data</code>	Sends data to the server.
<code>lrd_stmt</code>	Specifies an SQL statement to be processed.

Statement Correlating Functions

<code>lrd_save_col</code>	Saves the value of a table cell to a parameter.
<code>lrd_save_value</code>	Saves a place holder descriptor value to a parameter.

<code>lrd_save_ret_param</code>	Saves the value of a return-parameter to a parameter (CtLib only).
<code>lrd_save_last_rowid</code>	Saves the last rowid to a parameter. (Oracle)

Variable Handling Functions

<code>lrd_assign</code>	Assigns a null-terminated string to a variable.
<code>lrd_assign_ext</code>	Assigns a storage area to a variable.
<code>lrd_assign_literal</code>	Assigns a literal string (containing null-characters) to a variable.
<code>lrd_assign_bind</code>	Assigns a null-terminated string to a variable and binds it to a place holder.
<code>lrd_assign_bind_ext</code>	Assigns a storage area value to a variable and binds it to a place holder.
<code>lrd_assign_bind_literal</code>	Assigns a literal string (containing null-characters) to a variable and binds it to a place holder.
<code>lrd_to_printable</code>	Converts a variable to a printable string.

Siebel Functions

<code>lrd_siebel_incr</code>	Increments a string by a specified value.
<code>lrd_siebel_str2num</code>	Converts a base 36 string to a base 10 number.
<code>SiebelPostSave_x</code>	Saves the future values of Siebel parameters.
<code>SiebelPreSave_x</code>	Indicates the parameters necessary for correlation.

Oracle 8 Functions

VuGen provides partial support for Oracle 8.x. All database actions that were recorded in previous versions of Oracle are recorded. In many instances, the recorded function is specific for Oracle 8.x. For example for a fetch operation, instead of `lrd_fetch`, VuGen records `lrd_ora8_fetch`.

<code>lrd_attr_set</code>	Sets an attribute for an LRDDBI handle.
<code>lrd_attr_set_from_handle</code>	Sets an attribute using an LRDDBI handle pointer.
<code>lrd_attr_set_literal</code>	Sets an LRDDBI handle attribute using a literal string.
<code>lrd_env_init</code>	Allocates and initializes an LRDDBI handle.
<code>lrd_handle_alloc</code>	Explicitly allocates and initializes an LRDDBI handle.
<code>lrd_handle_free</code>	Explicitly frees an LRDDBI handle.
<code>lrd_initialize_db</code>	Initializes the database process environment.
<code>lrd_logoff</code>	Terminates a simple database session.
<code>lrd_logon</code>	Begins a simple database session.
<code>lrd_logon_ext</code>	Begins a simple database session (extended).
<code>lrd_ora8_attr_set</code>	Sets an attribute for an LRDDBI handle—shorthand.
<code>lrd_ora8_attr_set_from_handle</code>	Sets an attribute using an LRDDBI handle pointer.
<code>lrd_ora8_attr_set_literal</code>	Sets an LRDDBI handle attribute using a literal string—shorthand.
<code>lrd_ora8_bind_col</code>	Binds a host variable to an output column.
<code>lrd_ora8_bind_placeholder</code>	Binds a host variable to a placeholder.
<code>lrd_ora8_commit</code>	Commits the current transaction for an Oracle 8.x client.
<code>lrd_ora8_exec</code>	Executes an SQL statement in Oracle 8.x.
<code>lrd_ora8_fetch</code>	Fetches the next row in the result set.

lrd_ora8_handle_alloc	Explicitly allocates and initializes an LRDDBI handle—shorthand.
lrd_ora8_rollback	Rolls back the current transaction for an Oracle 8.x client.
lrd_ora8_save_col	Saves the value of a table cell to a parameter.
lrd_ora8_stmt	Prepares a null-terminated SQL statement for execution.
lrd_ora8_stmt_ext	Prepares an SQL statement with null characters for execution.
lrd_ora8_stmt_literal	Prepares a literal SQL statement string for execution.
lrd_server_attach	Creates an access path to a data source for database operations
lrd_server_detach	Deletes an access path to a data source for database operations
lrd_session_begin	Creates and begins a user session for a server.
lrd_session_end	Terminates a user session for a server.

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.

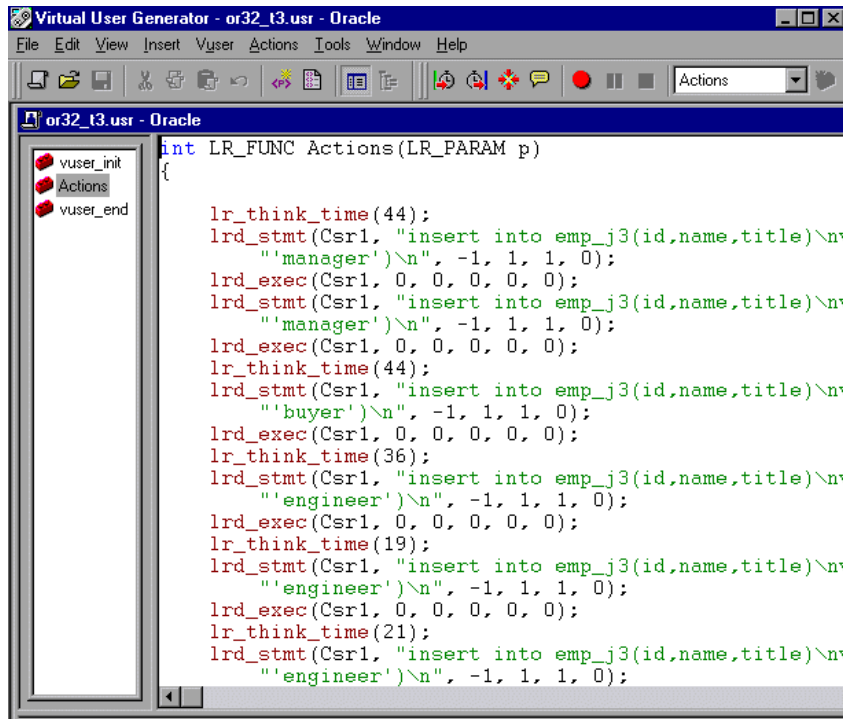
In the following script, VuGen recorded the actions of an operator who opened a connection to an Oracle server and then performed a query requesting the local settings.

```
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);
```

Grids

The data returned by a database 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.

In the following example, VuGen displays a grid for a query executed on an employee database. The query retrieves the name, ID and title for all employees with the title *engineer*.



The grid columns are adjustable in width. You can scroll up to 100 rows using the scroll bar.

To show or hide the grid select **View > Data Grids**.

Row Information

VuGen generates an `lrd_fetch` function for each SQL query.

```
lrd_fetch(Csrl, -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 always 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**, refer to the *Online Function Reference* (**Help > Function Reference**).

Evaluating Error Codes

When a Vuser executes an LRD function, the function always 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, refer to 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:

- ▶ globally—to the entire script, or to a segment of the script
- ▶ locally—to a specific function only

Modifying Error Handling Globally

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, etc. 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.

Modifying Error Handling Locally

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 the following database statement fails (e.g., the table does not exist), then 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 parametrize 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.

18

Correlating Database Vuser Scripts

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 describes:

- Scanning a Script for Correlations
- Correlating a Known Value
- Database Correlation Functions
- Correlating Siebel Scripts

The following information only applies to Database (CtLib, DbLib, Informix, Oracle, and ODBC, DB2-CLI) Vuser scripts.

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 and using the results of one statement as input to another. The two primary reasons for correlating Database Vuser scripts are:

- duplicate values are illegal

Suppose your script creates a new employee record in your database and assigns a unique ID for each employee. The database requires that each record be unique and not duplicated. If you try to replay this script, it will fail because that employee ID was already created during the recording session and it cannot be duplicated.

To overcome this problem, you use correlation to capture the ID assigned to the new employee, and use it for the remainder of the database session. In addition, you could use parameterization to retrieve unique data for each employee. (see Chapter 7, “Defining Parameters”).

- to simplify or optimize your code

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

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script to ensure 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 yourscript.

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 with automatic correlation:

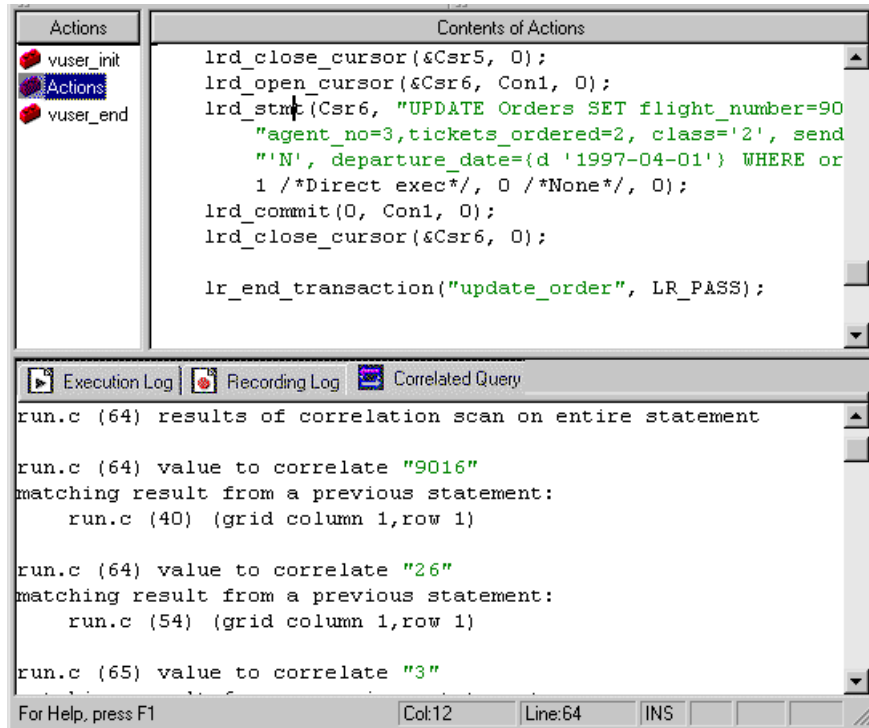
- 1 Open the Output window.

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

2 Select Vuser > Scan for Correlations.

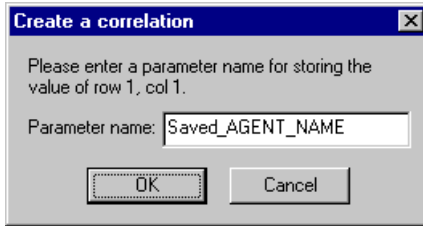
VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.

In the following example, in the `lrd_stmt` (Csr6, "UPDATE...") statement, VuGen detected two values to correlate.



- 3 In the Correlated Query tab, double-click on the result you want to correlate. This is located on the third line of the text 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 choose **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 (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**) which saves the result to a parameter.
- 6 Click **Yes** to confirm the correlation.
A message appears asking if you want to search for all occurrences of the value in the script.
- 7 Click **No** to replace only the value in the selected statement.
- 8 To search for additional occurrences click **Yes**. A Search and Replace dialog box opens.
- 9 Confirm any replacements, including your original statement. Close the Search and Replace dialog box
VuGen replaces the statement value with a reference to the parameter. Note that if you choose to 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.

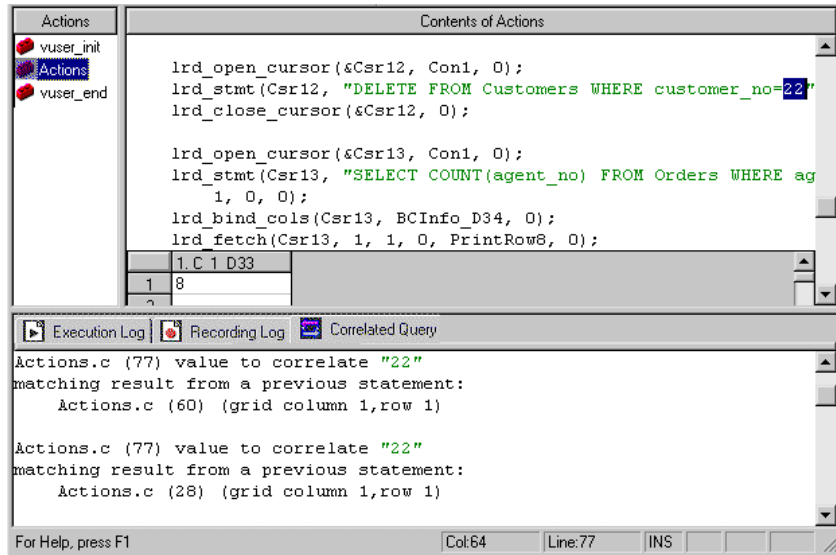
To correlate a specific value:

- 1 Locate the value you want to correlate and select the value without the quotation marks.

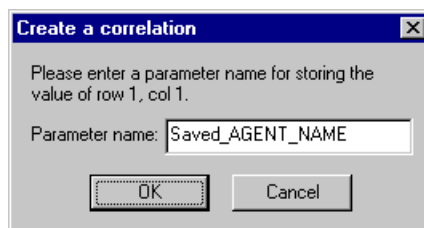
2 Choose **Vuser > Scan for Correlations (at cursor)**.

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.

In the following example, VuGen found two possible matching result values to correlate to "22".



- In the Correlated Query tab, double-click the result you want to correlate. This is located on the third 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.
- In the grid, select the value to correlate, and choose **Vuser > Create Correlation**. VuGen prompts you to enter a parameter name.



- 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**) which saves the result to a parameter.
- 6 Click **Yes** to confirm the correlation.

A message appears asking if you want to search for all occurrences of the value in the script.
- 7 Click **No** to replace only the value in the selected statement.
- 8 To search for additional occurrences, click **Yes**. The Search and Replace dialog box opens.
- 9 Confirm any replacements, including your original statement. Close the Search and Replace dialog box.

VuGen replaces the statement value with a reference to the parameter. Note that if you choose to undo the correlation, VuGen 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, etc.) 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_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, refer to the *Online Function Reference*.

Correlating Siebel Scripts

In Siebel there are special arrays that cannot be correlated using the standard correlation rules. Before saving the dynamic values to a parameter, you need to pre-process the value. In addition, you do not replace all instances of the value—only specific ones.

The Callback

When VuGen detects a match using the boundaries as criteria, it performs a callback to the parameter name and its value. The callback parses the array and saves each of its arguments to a separate parameter. The callbacks are defined in Siebel's correlation rules.

The **web_reg_save_param** functions added by those rules, have an additional parameter:

```
AutoCorrelationFunction=<callback_name>
```

The callback saves several parameters for each call to **web_reg_save_param**. The parameters that are saved by these callbacks are:

```
<callback_name>_1 = arg1
<callback_name>_2 = arg2
...
<callback_name>_rowid = <rowid>
```

```
web_reg_save_param("Siebel_Star_Array118",  
    "LB/IC='ValueArray'",  
    "RB/IC='",  
    "Ord=2",  
    "Search=Body",  
    "RelFrameId=1",  
    "AutoCorrelationFunction=flCorrelationCallbackParseStarArray",  
    LAST);
```

The parameters then can be replaced in the **web_submit_data** functions that follow.

Correlating 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 it, if it appears after the strings "SWEC=" or "SWEC"

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

Recording a Session

The recommended way to record the Siebel Application is:

- 1** Record the login in the *vuser_init* section
- 2** Record the Business Process to *Action1*
- 3** Record the logout in the *vuser_end* section
- 4** Set the following Recording Options:
 - Record in HTML mode with explicit URLs
 - Do not record out of context resources

Handling Error Messages

Correlation Parameter Not Found

The most common error message issued by VuGen when Siebel correlation fails, is "correlation parameter not found".

Solution 1: In `vuser_init`, search for all `web_submit_data` or `web_url` functions that include one of the following strings, and comment them out:

```
cdaprojects  
SWEFrame=top._swe  
SWEFrame%3Dtop._swe
```

Solution 2: Record the script in HTTP mode.

Time Stamp

To avoid time-stamp related errors. look for all instances of:

```
"Name=SWETS", "Value=1034965283662"
```

Replace the value with an empty string.

```
"Name=SWETS", "Value="
```


19

Developing DNS Vuser Scripts

VuGen allows you to emulate network activity by directly accessing a DNS server.

This chapter describes:

- ▶ Working with DNS Functions

The following information applies only to DNS Virtual User scripts.

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.

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 both of them. For instructions on specifying multiple protocols, see Chapter 3, "Recording with VuGen."

To create a script for the DNS protocol, you choose the Domain Resolution (DNS) protocol type in the Client/Server category. Since recording is not supported for DNS, you program the script with the appropriate DNS, ProTune, 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 session step on either a Windows or UNIX platform. For more information on integrating Vuser scripts in a session step, refer to the *ProTune Console 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;
}

```


20

Developing WinSock Vuser Scripts

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 describes:

- ▶ Getting Started with Windows Sockets Vuser Scripts
- ▶ Setting the Recording Options
- ▶ Using LRS Functions
- ▶ Switching Between Tree View and Script View

The following information applies to all protocols recorded on a Windows Sockets level.

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 such as LDAP. 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 `Irs` 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 both the tree view and the script view of the script. You can easily switch between the two views. See "Switching Between Tree View and Script View," on page 275 for more information.

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. Choose an application to record and set the recording options. Record typical operations on your application.

For details, see Chapter 3, "Recording with VuGen."

2 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, “Correlating Statements.”

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

6 Run the Vuser script from VuGen.

Save and run the Vuser script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

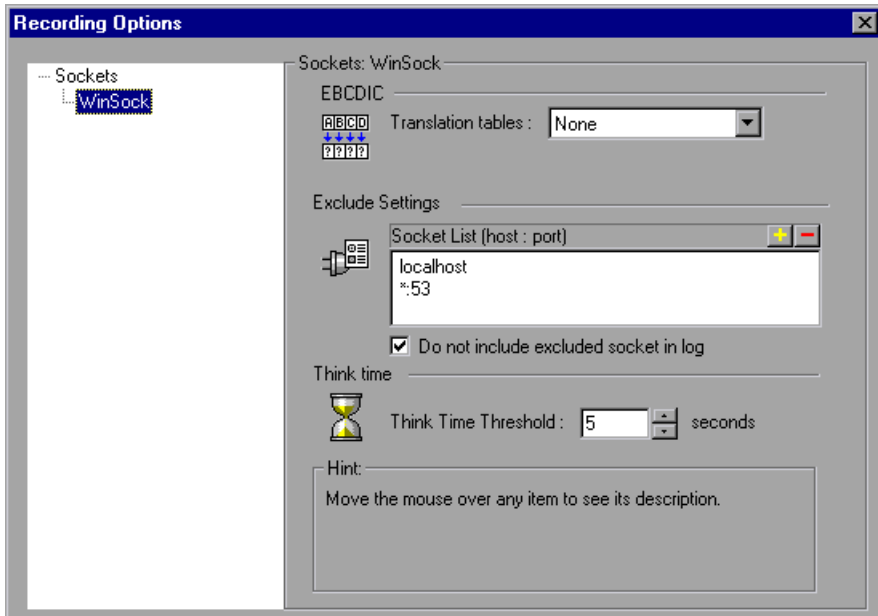
After you create a Windows Sockets Vuser script, you integrate it into a session step on either a Windows or UNIX platform. For more information on integrating Vuser scripts into a session step, refer to the *ProTune Console User's Guide*.

Setting the Recording Options

You can set the following recording options for your WinSock Vuser.

- ▶ Translation Table
- ▶ Socket Exclusion
- ▶ Think Time Threshold

To open the Recording Options dialog box, choose **Tools > Recording Options** or click the **Options** button in the Start Recording dialog box. VuGen displays the WinSock options.

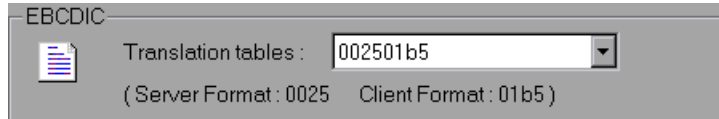


Note: These options also apply to the Web/WinSock dual-protocol.

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. Choose 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 choose 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 ProTune 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. It is recommended 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 **Irs** 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. For more information about the LRS functions, refer to the *Online Function Reference* (**Help > Function Reference**).

Socket Functions

	Irs_accept_connection	Accepts a connection on a listening socket.
	Irs_close_socket	Closes an open socket.
	Irs_create_socket	Initializes a socket.
	Irs_disable_socket	Disables an operation on a socket.
	Irs_exclude_socket	Excludes a socket during replay.
	Irs_get_socket_attrib	Gets a socket's attributes.
	Irs_get_socket_handler	Gets a socket handler for the specified socket.
	Irs_length_receive	Receives data from a buffer of a specified length.
	Irs_receive	Receives data from a socket.
	Irs_receive_ex	Receives data of a specific length from a datagram or stream socket.
	Irs_send	Sends data on a datagram or to a stream socket.
	Irs_set_receive_option	Sets a socket receive option.



lrs_set_socket_handler Sets a socket handler for the specified socket.



lrs_set_socket_options Sets a socket option.

Buffer Functions



lrs_free_buffer Frees the memory allocated for the buffer.



lrs_get_buffer_by_name Gets a buffer and its size from the data file.



lrs_get_last_received_buffer Gets the last buffer received on the socket and its size.



lrs_get_last_received_buffer_size Gets the size of the last buffer received on the socket.



lrs_get_received_buffer Gets the last received buffer or a part of it.



lrs_get_static_buffer Gets a static buffer or a part of it.



lrs_get_user_buffer Gets the contents of the user data for a socket.



lrs_get_user_buffer_size Gets the size of the user data for a socket.



lrs_set_send_buffer Specifies a buffer to send on a socket.

Environment Functions



lrs_cleanup Terminates the use of the Windows Sockets DLL.



lrs_startup Initializes the Windows Sockets DLL.

Correlating Statement Functions



lrs_save_param

Saves a static or received buffer (or part of it) to a parameter.



lrs_save_param_ex

Saves a user, static, or received buffer (or part of it) to a parameter.



lrs_save_searched_string

Searches for an occurrence of strings in a static or received buffer and saves a portion of the buffer, relative to the string occurrence, to a parameter.

Conversion Functions



lrs_ascii_to_ebcdic

Converts buffer data from ASCII format to EBCDIC format.



lrs_decimal_to_hex_string

Converts a decimal integer to a hexadecimal string.



lrs_ebcdic_to_ascii

Converts buffer data from EBCDIC format to ASCII format.



lrs_hex_string_to_int

Converts a hexadecimal string to an integer.

Timeout Functions



lrs_set_accept_timeout

Sets a timeout for accepting a socket.



lrs_set_connect_timeout

Sets a timeout for connecting to a socket.



lrs_set_rcv_timeout

Sets a timeout for receiving the initial expected data on a socket.



lrs_set_rcv_timeout2

Sets a timeout for receiving the expected data on a socket after a connection was established.



lrs_set_send_timeout

Sets a timeout for sending data on a socket.

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.

Switching Between Tree View and Script View

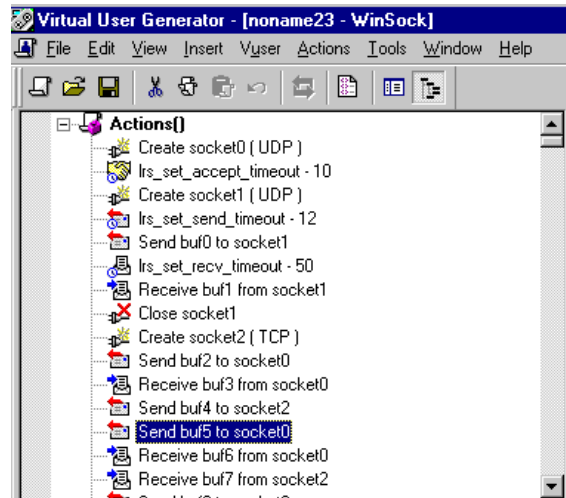
When viewing and editing a Windows Sockets Vuser script in VuGen, you choose between viewing the script in the icon-based tree view or the text-based script view.

To display the tree view of a WinSocket Vuser script:



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

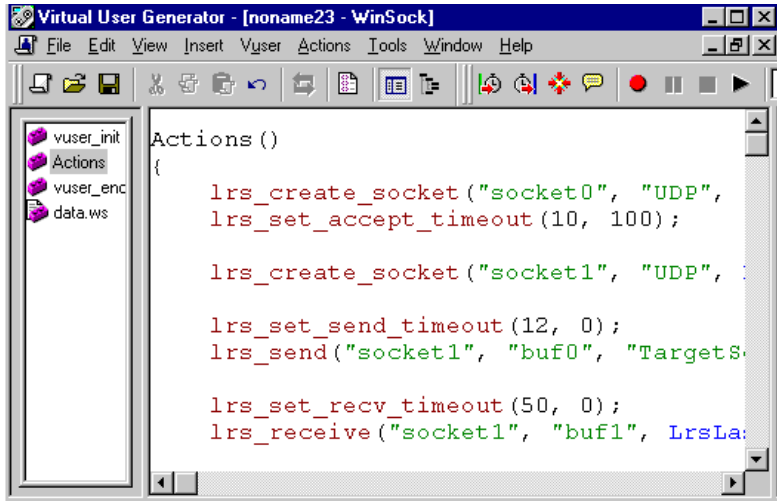
The Actions section of the Vuser script is displayed in the tree view. If you are already in this view, the menu item is disabled.



To display the script view:



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



After creating a script, you can view the data as a snapshot or as a raw data file. For details, see Chapter 21, “Working with Window Sockets Data.”

21

Working with Window Sockets Data

After you record a session in the Windows Socket protocol you can view and manipulate the data.

This chapter describes:

- ▶ Viewing Data in the Snapshot Window
- ▶ Navigating Through the Data
- ▶ Modifying Buffer Data
- ▶ Modifying Buffer Names
- ▶ Understanding the Data File Format
- ▶ Viewing Buffer Data in Hexadecimal format
- ▶ Setting the Display Format
- ▶ Debugging Tips
- ▶ Manually Correlating WinSock Scripts

The following information applies to all protocols recorded on a Windows Socket level.

About Working with Windows Socket Data

After you record an application using the VuGen, you have multiple data buffers containing the data.

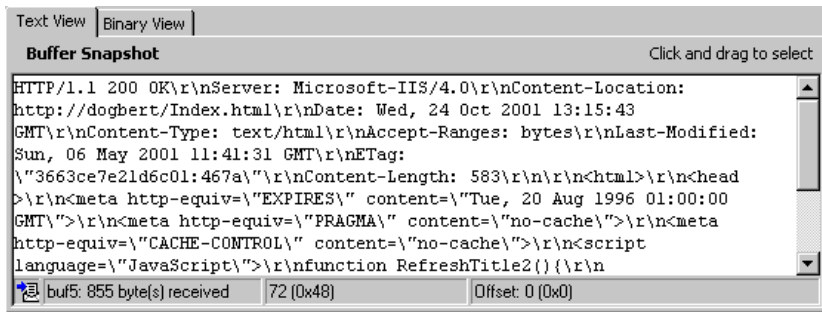
When you view the WinSocket script in tree view, VuGen provides a snapshot window which allows you to navigate within the data buffers and modify the data.

When working in script view, you can view the raw data in the *data.ws* file. For more information, see See “Viewing Windows Socket Data in Script View” on page 289.

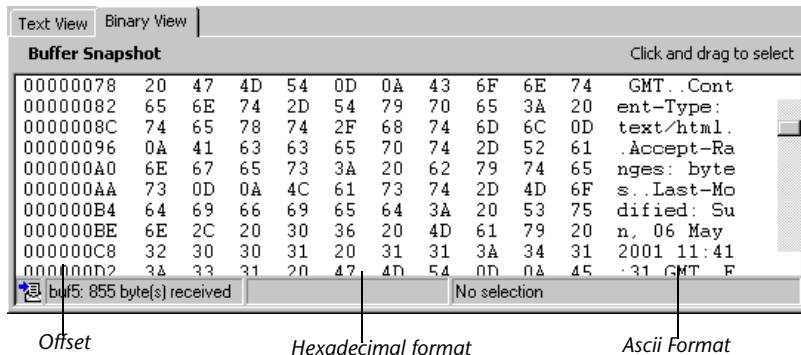
Viewing Data in the Snapshot Window

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.

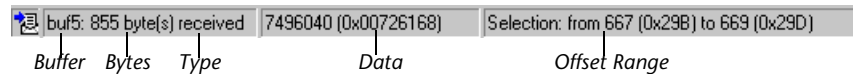


The binary view shows the data in hexadecimal representation. The left column shows the offset of the first character in that row. The middle column displays the hexadecimal values of the data. The right column shows the data in ASCII format.



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.



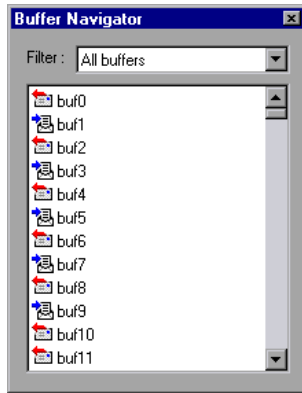
Navigating Through the Data

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, choose **View > Buffer Navigator**. To close the navigator, click the X in the top right corner of the navigator dialog box.

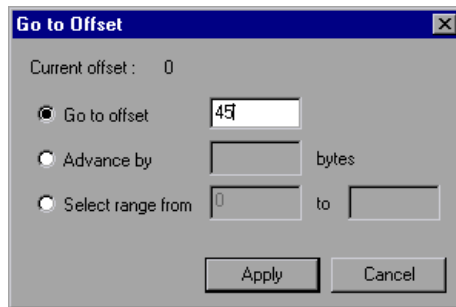
Note that you can also navigate between buffers by clicking on the buffer step in the left pane's tree view. The advantages of the buffer navigator are that it is a floating window with filtering capabilities.

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:

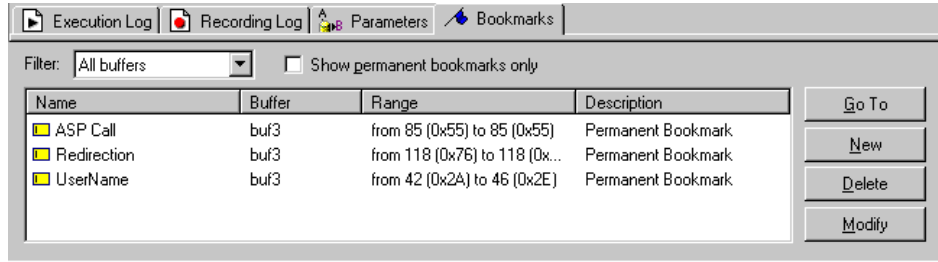
- 1 Click within the snapshot window. Then select **Go to offset** from the right-click menu. The Go to offset dialog box opens.



- 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 the binary view the bookmark block is marked in red. When you place your cursor over the bookmark in the buffer, a popup text box indicates 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 always 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, choose **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 choose *Selected buffer only* in the **Filter** box.
- 8 To delete a bookmark, select it in the **Bookmarks** tab and click **Delete**.

Modifying Buffer Data

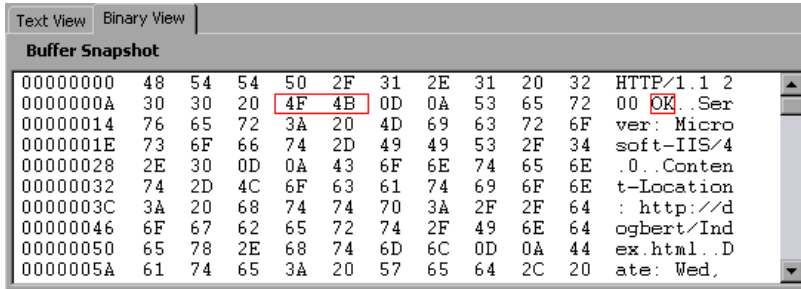
In tree view, VuGen provides several tools that allow you to modify the data by deleting, changing or adding to the existing data.

- Editing Data
- Parameterizing Data

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 choose **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 choose **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, choose **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 choose **Delete** from the right-click menu.

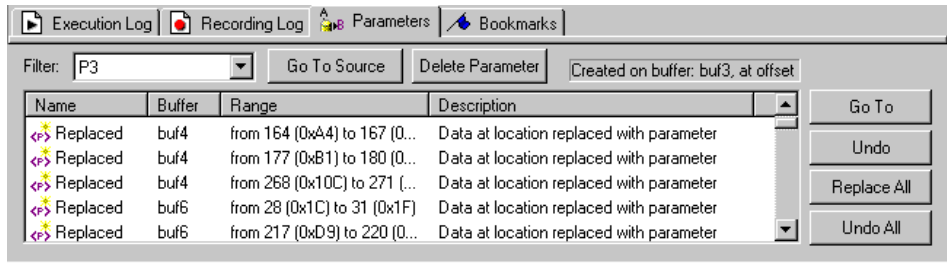
Parameterizing the Data

In tree view, VuGen lets you parameterize the data directly from the buffer snapshot view. You can specify a range of what to parameterize and you can specify borders. If you do not specify borders for the parameterized string, then VuGen inserts an `lrs_save_param` function into your script. If you specify borders, VuGen inserts `lrs_save_searched_string` into your script since this function allows you to specify boundary arguments. Note that the `lrs_save_param` and `lrs_save_searched_string` functions correlate the data. This means that it stores the data that is received, for use in a later point within the test. Since correlation stores the received data, it only applies to Receive buffers and not to Send buffers. The recommended procedure is to select a string of dynamic data within the Receive buffer that you want to parameterize. Use that same parameter in a subsequent Send buffer.

This type of correlation should not be confused with simple parameterization. Simple parameterization (**Insert > New Parameter**) only applies to data within Send buffers. You set up a parameter and assign it several values. VuGen uses the different values in each of the test runs or iterations. For more information, see Chapter 7, “Defining Parameters.” The next sections only 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 of the replaced parameter within the buffer.

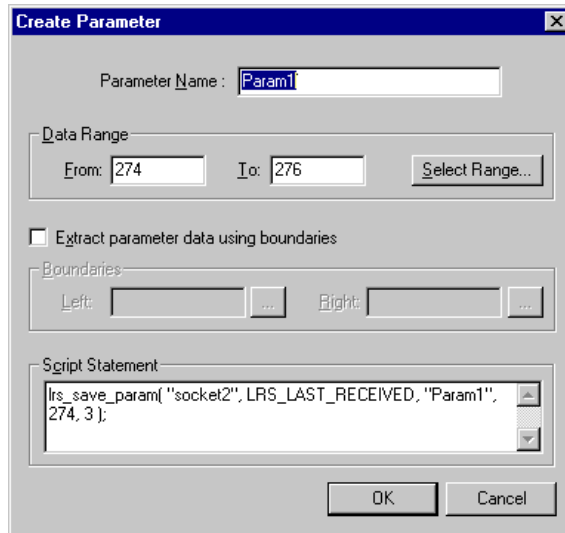
Deleting: You can delete any one of the parameters. When you delete a parameter, VuGen replaces the data with its original value and removes the parameterization function from the script.

Name: You can provide a name to each replacement.

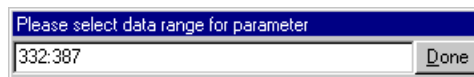
Undo Replacement: You can also undo one or more replacements displayed in the list.

To parameterize data from the snapshot window:

- 1 Select the data you want to parameterize and choose **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.



Choose a range in the buffer snapshot window and then click **Done**.

- 4 If the parameter data is not constant but its borders are consistent, you can specify a right and left boundary.
- 5 To specify boundaries:

Select the **Extract Parameter Data Using Boundaries** check box. VuGen changes the function in the **Script Statement** section from `lrs_save_param` to `lrs_save_searched_string`. Click **Done**.

Click the browse button adjacent to the **Left** box in the **Boundaries** section. A small dialog box opens, indicating your selection within the buffer. Select the boundaries within the buffer and click **Done**. Repeat this step for the right boundary.

- 6** Make the desired modifications to the arguments in the **Script Statement** section. For example you can add `_ex` to the `lrs_save_param` function to specify an encoding type. For more information about these functions see the *Online Function Reference*.
- 7** 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.
- 8** To jump to the original location of the parameter within its buffer, select it and click **Go To Source**.
- 9** To jump to the buffer location of the selected replacement, select it and click **Go To**.
- 10** To delete an entire parameter, choose the parameter in the **Filter** box and click **Delete Parameter**.
- 11** 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**.
- 12** When you undo specific replacements, the label changes from *Replaced* to *Found*. To reapply the parameterization rule to those that were undone, click **Replace** or **Replace All**.
- 13** To delete an entire parameter and undo all the replacements, select the parameter in the **Filter** box and click **Delete Parameters**.
- 14** Choose **Vuser > Parameter List** to assign data to the parameters.

Modifying Buffer Names

You can modify the name of a buffer using the Script view of the `data.ws` file. If you modify a buffer name after recording, this will affect the replay of the Vuser script. You can view the contents of the renamed buffer in the Script view or in Tree view using the Buffer Navigator.

If you created bookmarks in the buffer and it is not longer available, VuGen prompts you to delete the bookmarks within the buffer in which they were defined.

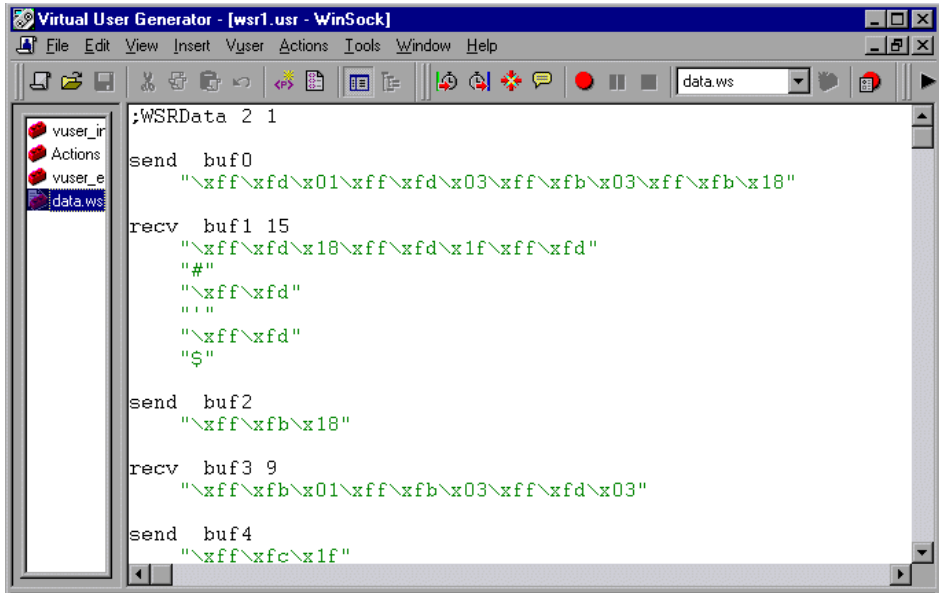
If you created parameters in the buffer and it is not longer available, VuGen prompts you to delete the parameters from the buffer in which they were defined. When you delete the parameter, all replacements are undone, even those in other buffers.

When you view the renamed buffer in the Buffer Navigator, VuGen warns you that parameter creation will be disabled within that buffer.

Viewing Windows Socket Data in Script View

When you use VuGen to create a Windows Sockets Vuser script, your actions are recorded into the three sections of the script: *vuser_init*, *Actions*, and *vuser_end*. In addition to the Vuser script, VuGen also creates a data file, *data.ws* that contains the data that was transmitted or received during the recording session. You can use VuGen to view the contents of the data file by selecting *data.ws* in the Data Files box of the main VuGen window.

The option to view a data file is available by default for Windows Sockets scripts. Note that you can only view the data in script view.



Several LRS functions, such as **lrs_receive** and **lrs_send**, handle the actual data that is transferred between servers and clients. The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the Vuser script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file, *data.ws*, contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRS functions use the buffer descriptors to access the data.

The descriptors have one of the following formats:

```
rcv bufindex number of bytes received
send bufindex
```

The buffer index begins with 0 (zero), and all subsequent buffers are numbered sequentially (1,2,3 etc.) regardless of whether they are send or receive buffers.

In the following example, an `lrs_receive` function was recorded during a Vuser session:

```
lrs_receive("socket1", "buf4", LrsLastArg)
```

In this example, `lrs_receive` handled data that was received on `socket1`. The data was stored in the fifth receive record(buf4)—note that the index number is zero-based. The corresponding section of the `data.ws` file shows the buffer and its contents.

```
recv buf4 39
  "\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
  "\r\n"
  "\r\n"
  "SunOS UNIX (sunny)\r\n"
  "\r"
  "\x0"
  "\r\n"
  "\r"
  "\x0"
```

Understanding the Data File Format

The `data.ws` data file has the following format:

- File header
- A list of buffers and their contents

The file header includes an internal version number of the data file format. The current version is 2. If you try to access data from a data file with format version 1, an error is issued.

```
;WSRData 2 1
```

An identifier precedes each record, indicating whether the data was received or sent, followed by the buffer descriptor, and the number of bytes received (for `lrs_receive` only). The buffer descriptor contains a number identifying the buffer. For example,

```
recv buf5 25
```

indicates that the buffer contains data that was received. The record number is 5, indicating that this receive operation was the sixth data transfer (the index is zero based), and twenty-five bytes of data were received.

If your data is in ASCII format, then 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, refer to See “Setting the Recording Options” on page 268. The EBCDIC whose ASCII value (after translation) is printable, is displayed as an ASCII character. If the ASCII value corresponds to a non-printable character, then VuGen displays the original EBCDIC value.

```
recv buf6 39
"\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
"\r\n"
"SunOS UNIX (sunny)\r\n"
```

The following segment shows the header, descriptors, and data in a typical data file:

```
;WSRData 2 1

send buf0
  "\xff\xfd\x01\xff\xfd\x03\xff\xfb\x03\xff\xfb\x18"

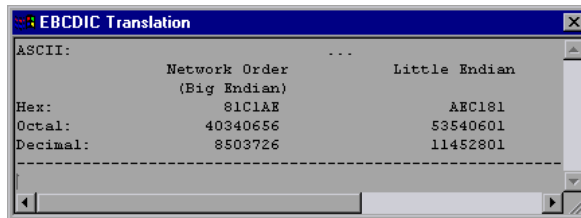
recv buf1 15
  "\xff\xfd\x18\xff\xfd\x1f\xff\xfd"
  "#"
  "\xff\xfd"
  ""
  "\xff\xfd"
  "$"

send buf2
  "\xff\xfb\x18"
```

Viewing Buffer Data in Hexadecimal format

VuGen contains a utility allowing you to view a segment of data, displaying it in hexadecimal and ASCII format, while indicating the offset of the data.

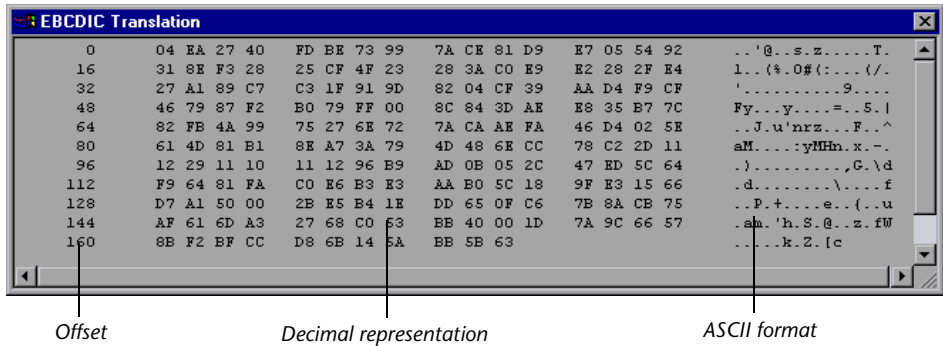
To display the data in the viewer window, select the data and press F7. If the selected text is less than four characters, VuGen displays the data in *short format*, showing the hexadecimal, decimal and octal representations.



You can customize the short format in the *conv_frm.dat* file as described in "Setting the Display Format" on page 295.

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 295.

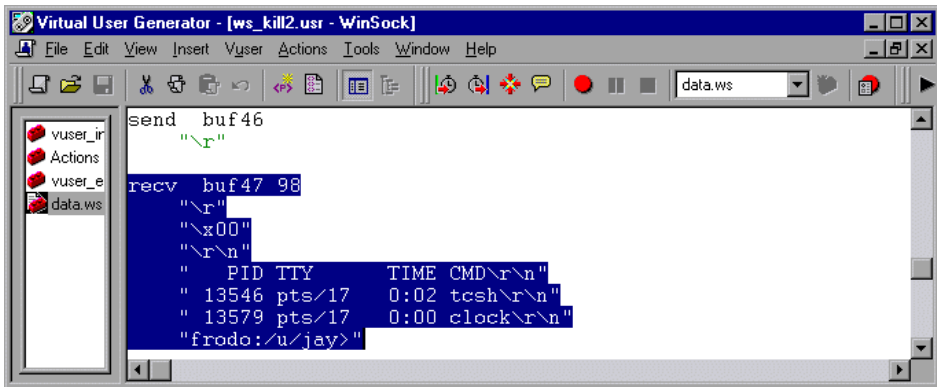
In the default format, the first column displays the character offsets from the beginning of the marked section. The second column displays the hexadecimal representation of the data. The third column shows the data in ASCII format. When displaying EBCDIC data, all non-printable ASCII characters (such as /n), are represented by dots.



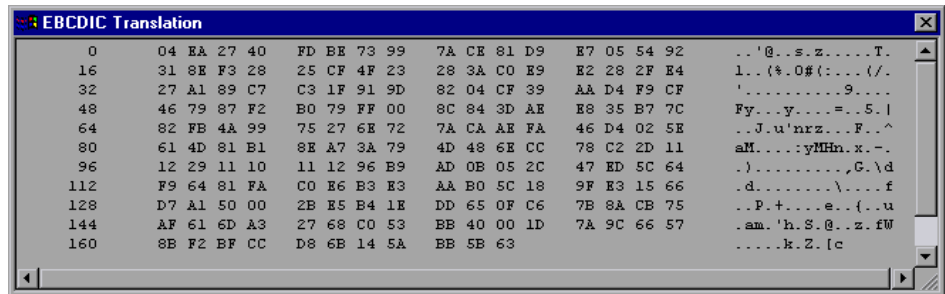
The F7 viewer utility is especially useful for parameterization. It allows you to determine the offset of the data that you want to save to a parameter.

To determine the offset of a specific character:

- 1 View *data.ws* and select the data from the beginning of the buffer.



- Press F7 to display the data and the character offsets. Since more than four characters were selected, the data is displayed in long format.



- 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.
- Use the offset value in the `lrs_save_param_ex` function in order to correlate the value of the process ID. For more information, see Chapter 8, “Correlating Statements.”

Setting the Display Format

You can specify how VuGen will display the buffer data in the viewer (F7) window. The `conv_frm.dat` file in the `lrun/dat` directory contains the following display parameters:

LongBufferFormat: The format used to display five or more characters. Use `nn` for offset, `XX` for the hex data, and `aa` for ASCII data.

LongBufferHeader: A header to precede each buffer in Long buffer format.

LongBufferFooter: A footer to follow each buffer in Long buffer format.

ShortBufferFormat: The format used to display four characters or less. You can use standard escape sequences and conversion characters.

The supported escape sequence characters are:

\a	Bell (alert)
\b	Backspace
\f	Formfeed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\?	Literal question mark
\ooo	ASCII character -octal

The supported conversion characters are:

%a	ASCII representation
%BX	Big Endian (Network Order) Hex
%BO	Big Endian (Network Order) Octal
%BD	Big Endian (Network Order) Decimal
%LX	Little Endian Hex
%LO	Little Endian Octal
%LD	Little Endian Decimal

AnyBufferHeader: A header to precede each buffer.

AnyBufferFooter: A footer to follow each buffer.

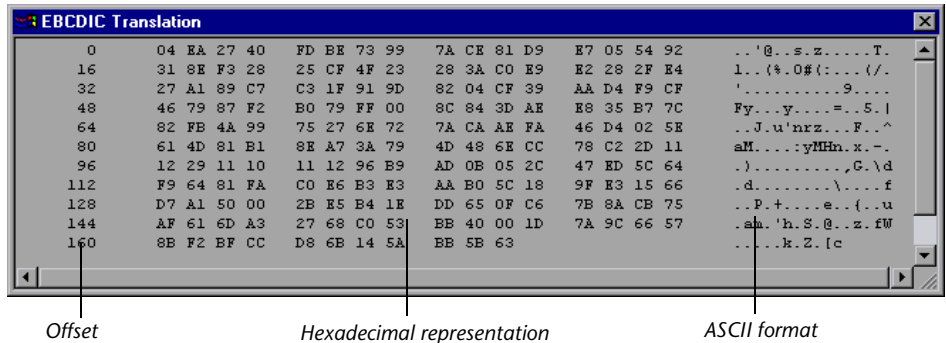
NonPrintableChar: The character with which to represent non-printable ASCII characters.

PrintAllAscii: Set to 1 to force the printing of non-printable ASCII characters.

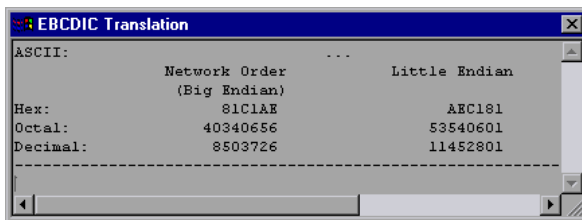
In the default settings, long and short formats are set, and a dot is specified for non-printable characters.

```
[BufferFormats]
LongBufferFormat=nnnnnnnnn  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\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. Choose **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.mercury.c\r\n
=====
The received buffer is:
=====
\r\n Last login: Thu Sep 10 11:19:50 from dolphin.mercury.c\r\n
```

However, if there is a very large discrepancy between the size of the Expected and Received buffers, this could indicate a problem with your system. Check the data in the corresponding buffer for discrepancies.

In order for you to determine whether or not the mismatch is significant, you must thoroughly understand your application.

Manually Correlating WinSock Scripts

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

You can manually correlate a Vuser script using the correlation functions that save the dynamic values to a parameter. The **lrs_save_param** and **lrs_save_param_ex** functions let you save data to a parameter based on the offset of the data in the received buffer. An advanced correlation function **lrs_save_searched_string** lets you designate the data by specifying its boundaries and indicating which occurrence of the matched pattern to save to a parameter. The following example describes correlation using **lrs_save_param_ex**. For information about using other correlation functions, see the *Online Function Reference*.

To correlate the WinSock Vuser statements:

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

```
lrs_save_param_ex (socket, type, buffer, offset, length, encoding, parameter);
```

- 2 Reference the parameter.

View the buffer contents by selecting the *data.ws* file in the Data Files box of the main VuGen window. Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in angle brackets (<>).

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

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

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

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

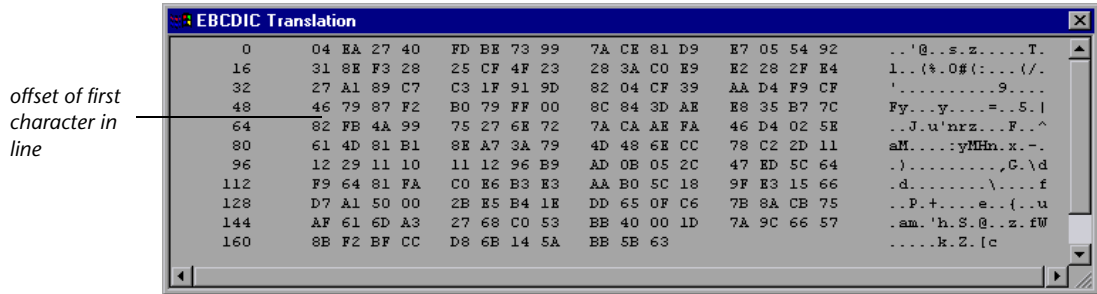
- 3 In the *data.ws* file, determine the buffer in which the data was received, *buf47*.

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

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

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

- 5 Determine the offset and length of the data string to save. Highlight the entire buffer and press F7. The offset of the *PID* is 11 and its length is 5 bytes. For additional information about displaying the data, refer to See “Understanding the Data File Format” on page 291.



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

```
lrs_receive("socket1", "buf79", LrsLastArg);
lrs_save_param("socket1", "user", buf47, 11, 5, ascii, param1);
lr_output_message ("param1: %s", lr_eval_string("<param1>"));
lr_think_time(10);
lrs_send("socket1", "buf80", LrsLastArg);
```

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

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

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

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


Part VI

Custom

22

Creating Custom Vuser Scripts

In addition to recording a session, you can create a custom Vuser script. You can use both ProTune API functions and standard C, Java, VB, VBScript, or Javascript code.

This chapter describes:

- C Vusers
- Java Vusers
- VB Vusers
- VBScript Vusers
- JavaScript Vusers

The following information applies to all custom Vuser scripts: C, JavaScript, Java, VB and VBScript.

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 ProTune API or standard programming functions. ProTune 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 session step.

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 ProTune libraries. For more information, see Chapter 62, "Creating Vuser Scripts in Visual Studio."

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, choose **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 refer to the *Online Function Reference* (**Help > Function Reference**) for a C reference with syntax and examples of commonly used C functions.

Guidelines for Using C Functions

All standard ANSI-C conventions apply to C Vuser scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other C programs. You declare and define variables using ANSI C conventions.

The C Interpreter that is used to run Vuser scripts accepts the standard ANSI C language. It does not support any Microsoft extensions to ANSI C.

Before you add any C functions to a Vuser script, note the following limitations:

- A Vuser script cannot pass the address of one of its functions as a callback to a library function.
- The *stdargs*, *longjmp*, and *alloca* functions are not supported in Vuser scripts.
- Vuser scripts do not support structure arguments or return types. Pointers to structures are supported.

- In Vuser scripts, string literals are read-only. Any attempt to write to a string literal generates an access violation.
- C Functions that do not return int, must be casted. For example,
`extern char * strtok();`

Calling libc Functions

In a Vuser script, you can call *libc* functions. However, since the interpreter that is used to run Vuser scripts does not support any Microsoft extensions to ANSI C, you cannot use Microsoft's include files. You can either write your own prototypes when necessary, or ask Mercury Interactive 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, choose **Java Vuser** from the **Custom** category, in the New Virtual User dialog box. VuGen creates an empty Java script:

```
import Irapl.Ir;  
  
public class Actions  
{  
  
    public int init() {  
        return 0;  
    }  
  
    public int action() {  
        return 0;  
    }  
  
    public int end() {  
        return 0;  
    }  
}
```

Note that for Java type Vusers, you can only edit the *Actions* class. Within the *Actions* class, there are three methods: *init*, *action*, and *end*. Place initialization code in the *init* method, business processes in the *actions* method, and cleanup code in the *end* method.

You can also use Java Vuser functions in Corba-Java and RMI-Java Vuser scripts.

VB Vusers

You can create an empty Visual Basic Vuser Script, in which you can place and Visual Basic code. This script type lets you incorporate your Visual Basic application into ProTune. To create an empty VB Vuser script, choose **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 ProTune and the VB application.

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 ProTune. To create an empty VBScript Vuser script, choose **VB Script Vuser** from the **Custom** category, in the New Virtual User dialog box. VuGen creates an empty VBScript Vuser script:

```
Public Function Actions()

    "TO DO: Place your action code here

Actions = lr.PASS
End Function
```

VuGen creates three sections, *vuser_init*, *action*, and *vuser_end*. Each of these sections contain a VBScript function—*Init*, *Actions*, and *Terminate* respectively. You place your code within these functions, as indicated by the *TO DO* comment.

An additional section that is viewable from VuGen, is the *global.vbs* file, which creates the objects for ProTune and the VB Script. For example, the following code creates the standard ProTune object:

```
Set pt = CreateObject("ProTune.PTApr")
```

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 ProTune. To create an empty JavaScript Vuser script, choose **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 ProTune and the JavaScript. For example, the following code creates the standard ProTune object:

```
var pt = new ActiveXObject("ProTune.PTApr")
```


23

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 chapter for Corba-Java, RMI-Java, EJB, or Jacada type protocols.

This chapter describes how to work with a *Java* Vuser to program a Vuser script in Java:

- Creating a Java Vuser
- Editing a Java Vuser Script
- VuGen's Java API
- Working with Java Vuser Functions
- Setting your Java Environment
- Running Java Vuser Scripts
- Compiling and Running a Script as Part of a Package
- Programming Tips

The following information applies to Java, EJB Testing, Corba-Java, RMI-Java, and Jacada Vuser scripts.

About Programming Java Scripts

To prepare Vuser scripts using Java code, use the *Java*, *Corba-Java*, or *RMI-Java* type Vusers. These Vuser types support 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 "//".

The chapters on Corba, RMI, EJB, and Jacada Vusers explain how to create a script through recording. 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 ProTune 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, ensure 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. Once you verify that the script is functional, you incorporate it into a ProTune session step.

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 Choose **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...
<i>init</i>	a login to a server	the Vuser is initialized (loaded)
<i>action</i>	client activity	the Vuser is in "Running" status
<i>end</i>	a log off procedure	the Vuser finishes or is stopped

Init Method

Place all the login procedures and one-time configuration settings in the *init* method. The *init* method is only executed once—when the Vuser begins running the script. The following sample *init* method initializes an applet.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;

// Public function: init
public int init() throws Throwable {

    // Initialize Orb instance...
    MApplet mapplet = new MApplet("http://chaos/classes/", null);
    orb = org.omg.CORBA.ORB.init(mapplet, null);
    ...
}
```

Action Method

Place all Vuser actions in the *action* method. The *action* method is executed according to the number of iterations you set in the runtime settings. For more information on the iteration settings, see Chapter 9, “Configuring Run-Time Settings.” The following sample *action* method retrieves and prints the Vuser ID.

```
public int action() {
    lr.message("vuser: " + lr.get_vuser_id() + " xxx");
    return 0;
}
```

End Method

In the *end* method, place the code you want ProTune to execute at the end of the session step 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;
}
```

VuGen's Java API

ProTune provides a specific Java API for accessing Vuser functions. These functions are all static methods of the `lrapi.lr` class. The following section lists ProTune's Java Vuser 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.

Transaction Functions

<code>lr.start_transaction</code>	Marks the beginning of a transaction.
<code>lr.end_transaction</code>	Marks the end of a transaction.

Command Line Parsing Functions

<code>lr.get_attrib_double</code>	Retrieves a <i>double</i> type variable used on the script command line.
<code>lr.get_attrib_long</code>	Retrieves a <i>long</i> type variable used on the script command line.
<code>lr.get_attrib_string</code>	Retrieves a string used on the script command line.

Informational Functions

<code>lr.value_check</code>	Checks the value of a parameter.
<code>lr.user_data_point</code>	Records a user-defined data sample.
<code>lr.get_group_name</code>	Returns the name of the Vuser's group.

<code>lr.get_host_name</code>	Returns the name of the load generator executing the Vuser script.
<code>lr.get_master_host_name</code>	Returns the name of the machine running the ProTune Console.
<code>lr.get_object</code>	Captures a Java object and dumps it to a data file. (Corba-Java only)
<code>lr.get_scenario_id</code>	Returns the id of the current session step
<code>lr.get_vuser_id</code>	Returns the id of the current Vuser.

String Functions

<code>lr.deserialize</code>	Expands an object to represent its ASCII components.
<code>lr.eval_string</code>	Replaces a parameter with its current value.
<code>lr.eval_data</code>	Replaces a parameter with a byte value.
<code>lr.eval_int</code>	Replaces a parameter with an integer value.
<code>lr.eval_string</code>	Replaces a parameter with a string.
<code>lr.next_row</code>	Indicates to use the next row of data for the specified parameter.
<code>lr.save_data</code>	Saves a byte as a parameter.
<code>lr.save_int</code>	Saves an integer as a parameter.
<code>lr.save_string</code>	Saves a null-terminated string to a parameter.

Message Functions

<code>lr.debug_message</code>	Sends a debug message to the Output window.
<code>lr.enable_redirection</code>	Enables the redirection of standard messages and errors to a log file, as standard output and standard error.
<code>lr.error_message</code>	Sends an error message to the Vuser log file and Output window with location details.

<code>lr.get_debug_message</code>	Retrieves the current message class.
<code>lr.log_message</code>	Sends a message to the Vuser log file.
<code>lr.message</code>	Sends a message to a the Output window.
<code>lr.output_message</code>	Sends a message to the log file and Output window with location information.
<code>lr.redirect</code>	Redirects a string to a file.
<code>lr.set_debug_message</code>	Sets a debug message class.
<code>lr.vuser_status_message</code>	Sends a message to the Vuser Status area in the Console window.

Run-Time Functions

<code>lr.peek_events</code>	Indicates where a Vuser script can be paused.
<code>lr.rendezvous</code>	Sets a rendezvous point in a Vuser script.
<code>lr.think_time</code>	Pauses script execution to emulate the time a real user pauses to think between actions.

To use additional Java classes, import them at the beginning of the script as shown below. Remember to add the classes directory or relevant jar file to the classpath. Make sure that the additional classes are thread-safe and scalable.

```
import java.io.*;
import lrapi.*;

public class Actions
{
  ...
}
```

Working with Java Vuser Functions

You can use Java Vuser functions to enhance your scripts by:

- Inserting Transactions
- Inserting Rendezvous Points
- Obtaining Vuser Information
- Issuing Output Messages
- Emulating User Think Time
- Handling Command Line Arguments

Inserting Transactions

You define transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified requests. These requests can be simple or complex tasks. During and after the session step run, you can analyze the performance per transaction using ProTune's online monitor and graphs.

You can also specify a transaction status: lr.PASS or lr.FAIL. You can let ProTune 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

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 Console 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:

- 1 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 ProTune Console.
lr.get_scenario_id	Returns the id of the current session step.
lr.get_vuser_id	Returns the id of the current Vuser.

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

Issuing Output Messages

When you run a session step, the Console's Output window displays information about script execution. You can include statements in a Vuser script to send error and notification messages to the Console. The Console 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 “Message Functions,” on page 318 or refer to the *Online Function Reference* (**Help > Function Reference**).

You can instruct ProTune to redirect the Java standard output and standard error streams to the VuGen 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, refer to 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 ProTune handles think time functions, open the runtime settings dialog box. For more information, see Chapter 9, “Configuring Run-Time Settings.”

For more information about the `lr.think_time` function, refer to 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 can insert command line options after the script path in the ProTune Console. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

<code>lr.get_attrib_double</code>	Retrieves double precision floating point type arguments
<code>lr.get_attrib_long</code>	Retrieves long integer type arguments
<code>lr.get_attrib_string</code>	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, refer to the *Online Function Reference* (**Help > Function Reference**). For additional details on including arguments on a command line, refer to the *ProTune Console User's Guide*.

Setting your Java Environment

Before running your Java Vuser script, ensure 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*) and all of the ProTune classes (*protune/classes*).
- ▶ All classes used by the Java Vuser must be in the classpath—either set in the machine's `CLASSPATH` environment variable or in the **Java VM** tab in 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 just.do.it;

import Irapl.*;
public class Actions
{
    :
}
```

In the above example, Vugen, automatically creates the *just/do/it* directory hierarchy under the Vuser directory, and copies the *Actions.java* file to *just/do/it/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 in 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 Console (for scalability reasons), you need to ensure that all of the imported code is thread-safe.

Thread-safety is usually easy to implement, but harder to detect. A Java Vuser may run flawlessly under Vugen and under the Console 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 Irapl.*;
public class Actions
{
    private static int iteration_counter = 0;

    public int init() {
        return 0;
    }

    public int action() {
        iteration_counter++;
        return 0;
    }

    public int end() {
        lr.message("Number of Vuser iterations: "+iteration_counter);
        return 0;
    }
}
```

When you run one Vuser, the *iteration_counter* member accurately determines the number of iterations that were executed. When multiple Vusers run together as threads on a single virtual machine, the static class member *iteration_counter* is shared by all threads, resulting in an incorrect counting. The total number of all Vusers iterations is counted.

If code is known to be non thread-safe and you still want to import it into your script, you can run the Vusers as processes. For more information on running Vusers as threads or processes, see Chapter 9, “Configuring Run-Time Settings.”

When you run a simple Java Vuser script, it usually consists of a single thread—the main thread. Only the main thread can access the ProTune Java API. If a Java Vuser spawns secondary worker threads, using the ProTune API may cause unpredictable results. Therefore, it is recommended to use the ProTune Java API only in the main thread. Note that this limitation also affects the **lr.enable_redirection** function.

The following example illustrates where the 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;
}
```


Part VII

Distributed Component Protocols

24

Recording COM Vuser Scripts

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 using a Visual Basic add-in. For more information about the Visual Basic add-in, refer to Chapter 62, “Creating Vuser Scripts in Visual Studio.”

Chapter 25, “Understanding COM Vuser Scripts,” explains how VuGen COM scripts work and provides a brief function reference.

This chapter describes:

- COM Overview
- Getting Started with COM Vusers
- Selecting COM Objects to Record
- Setting COM Recording Options

The following information applies only to COM Vuser scripts.

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. For more information, see Chapter 4, “Selecting a Script Generation Language.”

COM Overview

This section provides an outline of COM technology. This should be enough to get you started with COM Vuser scripts. Refer to 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.

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. Choose an application to record and set the recording options. To set the script related recording options, see Chapter 4, “Selecting a Script Generation Language.” To set the COM specific options and filters, see the “Setting COM Recording Options” on page 338. Record typical operations using your application.

For details about recording, see Chapter 3, “Recording with VuGen.”

2 Refine the Object Filter.

Use the log file that was generated to refine your choice of objects to be recorded in the filter. See the following section, “Selecting COM objects to Record,” for details.

3 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a COM Vuser script, you integrate it into a session step on a Windows platform. For more information on integrating Vuser scripts in a session step, refer to your *ProTune Console User’s Guide*.

Selecting COM Objects to Record

The application you are testing may use a great many COM objects. 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 choose, try using the default filter.

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.

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

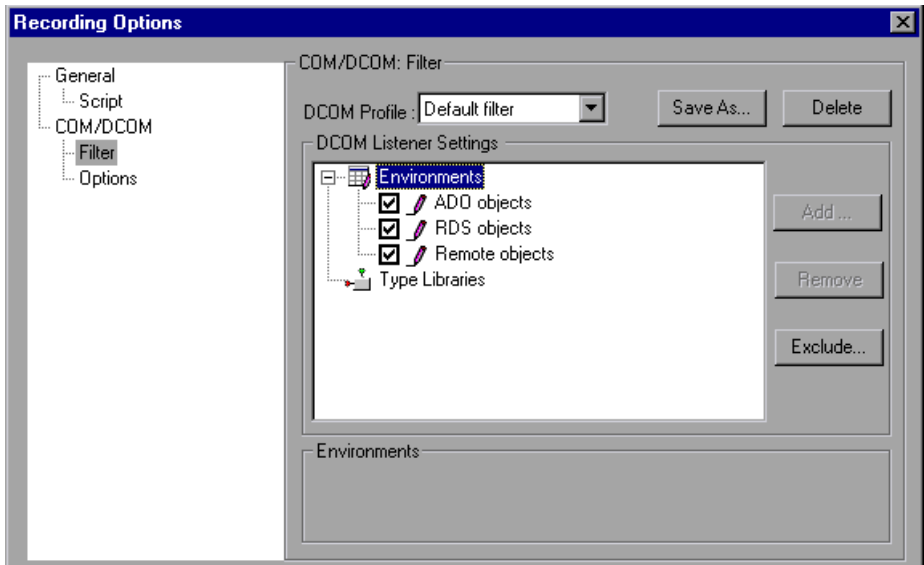
Use the COM Recording Options dialog box to set the filtering and COM scripting options. You use the online browser to locate type libraries, in the registry, file system, or the Microsoft Transaction Server (MTS).

Filtering Objects

The Filter options let you indicate which COM objects should be recorded by VuGen.

To tell VuGen which COM objects to record:

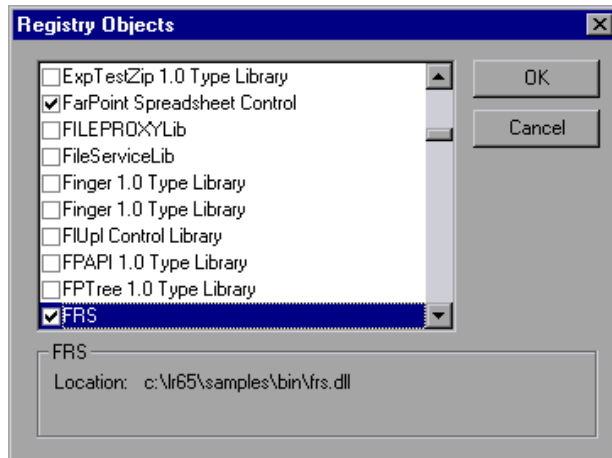
- 1 Choose **Tools > Recording Options** from the main menu or click **Options** in the Start Recording dialog box. A dialog box opens displaying the Recording Options tree. Select the **COM/DCOM:Filter** node.



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. You can clear the check box adjacent to **ADO**, **RDS** or **Remote objects** to exclude them from the filter.

- 2 All COM objects are represented in type libraries. You can add type libraries from the registry or file system such as *.tlb or *.dll. You can also add components from the Microsoft Transaction Server, if the computer has an MTS client installed. 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**.

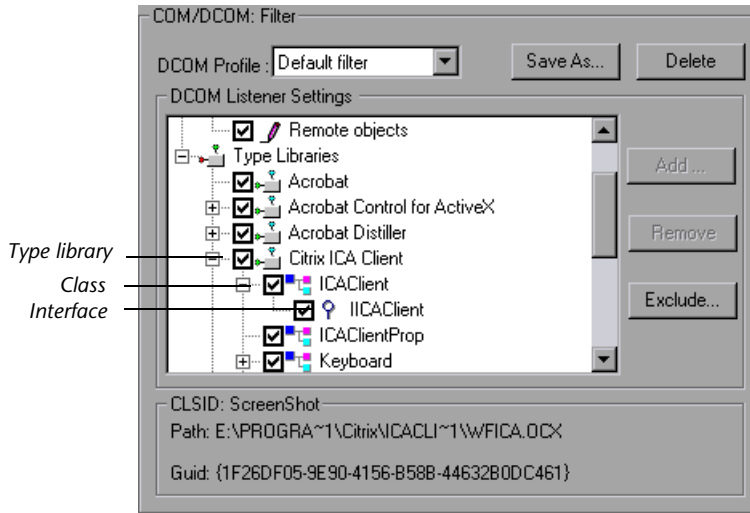
- 4 To add a type library from the file system, click **Add** and select **Browse file system**.

Select the desired file and click **OK**.

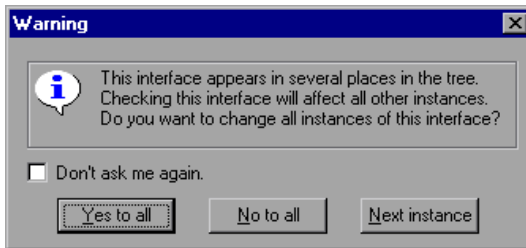
- 5 Once the type library appears in the list of Type Libraries, you can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

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.

Note that when you clear a check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

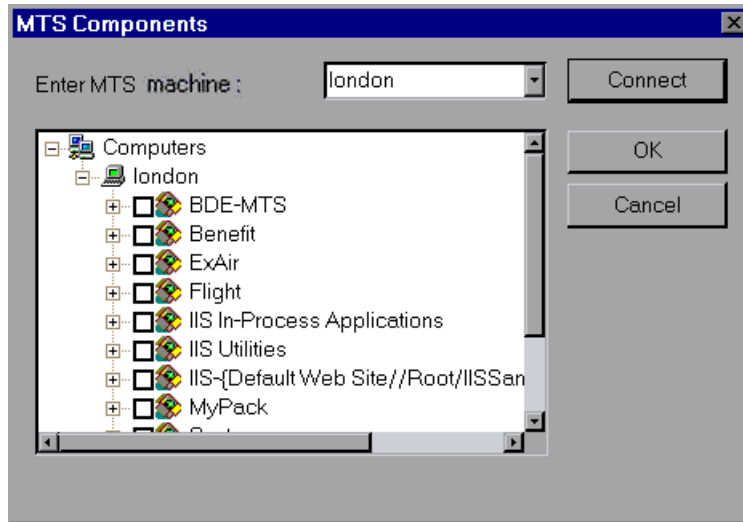


- An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the following warning:



If you check **Don't ask me again** and close the dialog, 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.

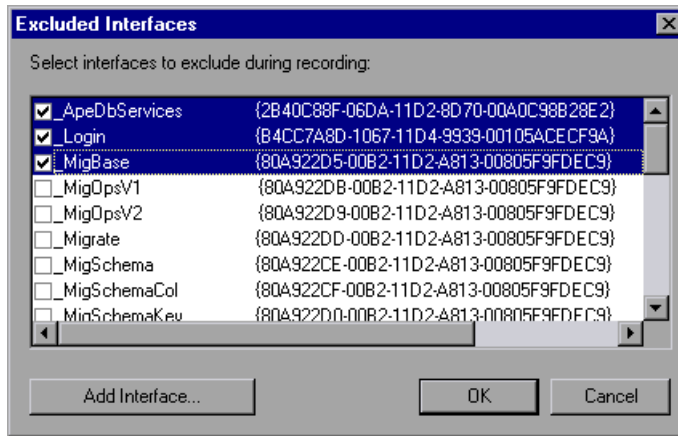


Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine.

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 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.

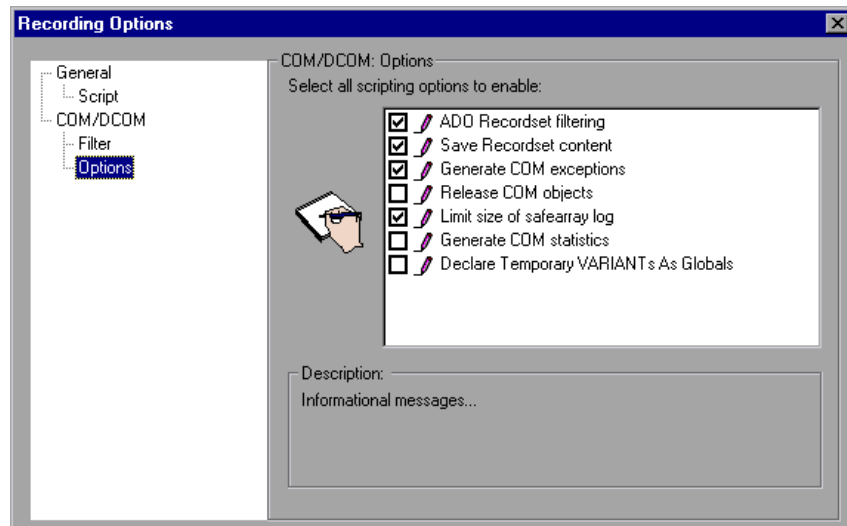
- 9 When you are finished making changes to an existing filter, click **OK** to save it and close the dialog box. Click **Save As** to save a **New filter**, or to save an existing filter under a new name. You can select saved filters in subsequent recordings. Default settings are given initially in 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.

To set COM/DCOM options:

- 1 Choose **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 in the check box adjacent to them. The following are the available options:

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)

Save Recordset content: Stores Recordset content as grids, to allow viewing of recordset in VuGen. (enabled by default)

Generate COM exceptions: Generate COM functions and methods that raised exceptions during recording. (enabled by default)

Release COM Objects: Record the releasing of COM objects when they are no longer in use. (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)

Generate COM statistics: Generate recording time performance statistics and summary information. (disabled by default)

Declare Temporary VARIANTS as Globals: Define temporary VARIANT types as Globals, not as local variables. (disabled by default)

When you finish selecting the desired options, click **OK** to save your settings and exit.

25

Understanding COM Vuser Scripts

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, refer to Chapter 24, “Recording COM Vuser Scripts.”

This chapter describes:

- ▶ Understanding VuGen COM Script Structure
- ▶ Examining Sample VuGen COM Scripts
- ▶ Scanning a Script for Correlations

The following information applies only to COM Vuser scripts.

About COM Vuser Scripts

When you record COM client communications, VuGen creates a script with calls to COM API functions and interface methods. In addition, you can program COM type conversion functions. Each function call has an **lrc** prefix, such as **lrc_CoCreateInstance** or **lrc_long**. This chapter provides an overview of COM API and type conversion calls. Refer to the *Online Function Reference* (**Help > Function Reference**), for syntax and examples of each function.

Calls to interface methods have the following names and syntax conventions:

lrc_<interface name>_<method name>(instance, ...);

Note that the **instance** is always the first parameter passed.

The vendors of the respective COM components usually supply documentation for the interface functions.

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

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.

Understanding VuGen COM Script Structure

VuGen COM scripts are structured in a special way to meet the needs of COM interfaces.

Interface Methods

Calls to interface methods have the following names and syntax conventions:

```
lrc_<interface name>_<method name>(instance,...);
```

Note that the **instance** is always the first parameter passed.

The vendors of the respective COM components usually supply documentation for the interface functions.

Interface Pointers

The interfaces.h file defines the interface pointers, as well as other variables, that will be used later on in the script. Each interface has an Interface ID (IID) which uniquely identifies the interface.

The format of the interface definition is:

```
<interface type>*<interface name> = 0; ///{<IID of the interface type>}"
```

In the following example, the interface type is **IDispatch**, the name of the interface instance is **IDispatch_0**, and the **IID** of **IDispatch** type is the long number string:

```
IDispatch* IDispatch_0= 0;/{00020400-0000-0000-C000-000000000046}"
```

Vuser Script Statements

The COM Vuser script consist of code that creates object instances, retrieves interface pointers and calls the interface methods. Each user action may generate one or more COM calls. Each COM call is coded by VuGen as a group of statements. Each such group is contained in a separate scope enclosed in braces. Several different statements prepare for the main call by assigning values and performing type conversions. For example, the group of calls needed to create an object may look like this:

```
{
  GUID pClsid = Irc_GUID("student.student.1");
  IUnknown * pUnkOuter = (IUnknown*)NULL;
  unsigned long dwClsContext = Irc_ulong("7");
  GUID riid = IID_IUnknown;
  Irc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid,
    (void**)&IUnknown_0, CHECK_HRES);
}
```

Error Checking

Each COM method or API call returns an error value. Vugen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

CHECK_HRES

This value is inserted if the function passed during recording and errors should be checked during replay.

DONT_CHECK_HRES

This value is inserted if the function failed during recording and errors should not be checked during replay.

Examining Sample VuGen COM Scripts

This section shows examples of how VuGen emulates a COM client application.

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. It returns a pointer to the **IUnknown** interface needed for the next stage:

```
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 returned 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** returns a pointer to the **Istudent** interface if the Istudent object has such an interface:

```
lrc_IUnknown_QueryInterface(IUnknown_0, &riid, (void**)&Istudent_0,
CHECK_HRES);
```

Using an Interface to Set Data

Here is an example of using the methods of the interface to set data. Suppose that in the application, the user is supposed to input a name. This activates a method for setting the name. VuGen records this in two statements. One statement is used for setting up the name string and the second one sets the name property.

To set up the entire function call:

- 1 First, VuGen sets a variable (**Prop Value**) equal to the string. The parameter is of type BSTR, a string type used in COM files:

```
BSTR PropValue = lrc_BSTR("John Smith");
```

In subsequent stages, you will probably parameterize this call, replacing “John Smith” with a parameter, so that different names are used each time the Vuser script is run.

- 2 Next, VuGen calls the **Put_Name** method of the **Istudent** interface to enter the name:

```
lrc_Istudent_put_name(Istudent_0, PropValue, CHECK_HRES);
```

Using an Interface to Return Data

Returning data from an application is different than entering the data, because you might want to store these values and use them as inputs in subsequent calls for parameterization.

This 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 = lrc_DispatchMethod1((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:

```

{
    GUID riid = IID_IDispatch;
    Irc_IOptional_QueryInterface(IOptional_0, &riid,
(void**)&IOptional_0, CHECK_HRES);
}
{
    VARIANT P1 = Irc_variant_short("47");
    VARIANT P2 = Irc_variant_short("37");
    VARIANT P3 = Irc_variant_date("3/19/1901");
    VARIANT var3 = Irc_variant_scode("4");
    Irc_DispatchMethod((IDispatch*)IOptional_0, "in_out_optional_args",
/*locale*/1024, &P1, &P2, OPTIONAL_ARGS, "#3", &P3, "var3", &var3,
LAST_ARG, CHECK_HRES);
}

```

The different **Irc_Dispatch** methods that use the **IDispatch** interface are detailed in the “LRC Function Reference” Section.

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 “LRC Function Reference” Section below. Here is an example. Previously, we showed how the **Irc_DispatchMethod1** call was used to retrieve an array of name strings:

```

VARIANT retVal = Irc_variant_empty();
retVal = Irc_DispatchMethod1((IDispatch*)IDispatch_0, "GetAgentsArray",
/*locale*/1033, LAST_ARG, CHECK_HRES);

```

Now we will show 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 Chapter 26, “Understanding COM Vuser Functions” or refer to the *Online Function Reference*.

Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script to ensure 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 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 Execution Log folder. 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, VuGen found several possible values to correlate in the `lrc_variant_BSTR` ("SELECT...") statement.

The screenshot shows the Virtual User Generator interface. The main editor displays a script snippet:

```
long Options = lrc_long ("-1");
VARIANT Source = lrc_variant_BSTR("SELECT * F
VARIANT ActiveConnection = lrc_variant_BSTR("
lrc_Recordset20_Open(Recordset20_4, Source, A
)
```

Below the script is a table with the following data:

	FLIGHT NUMBER	DEPARTURE INITIALS	DEPARTURE	DAY OF WEE
1	5709	DEN	Denver	Saturday
2	3636	DEN	Denver	Saturday
3				
4				

The **Correlated Query** tab at the bottom shows the following output:

```
vuser_init.c (101) (grid column 1, row 2)
vuser_init.c (164) value to correlate "Los Angeles"
matching result from a previous statement:
vuser_init.c (134) (grid column 1, row 2)
vuser_init.c (63) value to correlate "Alex"
matching result from a previous statement:
vuser_init.c (33) (grid column 1, row 1)
```

3 Correlate the value.

In the Correlated Query tab, double-click on the result you want to correlate. This is located on the third 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.

1 In the grid, select the value you want to correlate, and choose **Vuser > Create Correlation**. VuGen prompts you to enter a parameter name for the result value.

2 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.

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

A message appears asking if you want to search for all occurrences of the value in the script.

4 Click **No** to replace only the value in the selected statement.

5 To search for additional occurrences click **Yes**.

A Search and Replace dialog box opens. Confirm any replacements, including your original statement. After you replace all the desired values, click **Cancel** to close the Search and Replace dialog box.

VuGen replaces the statement value with a reference to the parameter. Note that if you choose to 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:

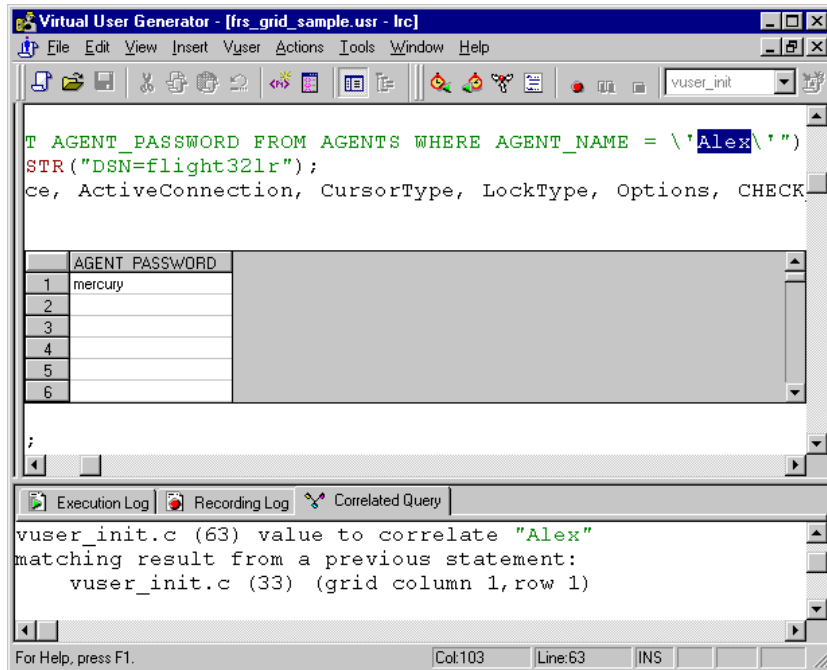
To correlate a specific value:

1 Locate the value you want to correlate and select the value without the quotation marks.

2 Choose **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.

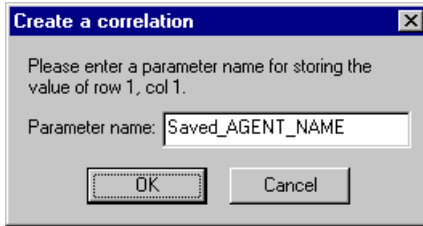
In the following example, VuGen found one matching result value to correlate to “Alex”.



In the Correlated Query tab, double-click on the result you want to correlate. This is located on the third 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.

- 3 In the grid, select the value you want to correlate and choose **Vuser > Create Correlation**. VuGen prompts you to enter a parameter name for the result value.



- 4 Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrc_save_<type>**) which saves the result to a parameter.

```
lrc_save_rs_param (Recordset20_0, 1, 1, 0, "Saved_AGENT_NAME");
```

- 5 Click **Yes** to confirm the correlation.

A message appears asking if you want to search for all occurrences of the value in the script.

- 6 Click **No** to replace only the value in the selected statement.

- 7 To search for additional occurrences click **Yes**.

A Search and Replace dialog box opens. Confirm any replacements, including your original statement. After you replace all the desired values, click **Cancel** to close the Search and Replace dialog box.

VuGen replaces the statement value with a reference to the parameter. Note that if you choose to cancel the correlation, VuGen also erases the statement created in the previous step.

26

Understanding COM Vuser Functions

The COM Vuser functions emulate the actions of a user running a COM application.

This chapter describes:

- Creating Instances
- IDispatch Interface Invoke Method
- Type Assignment Functions
- Variant Types
- Assignment from Reference to Variant
- Parameterization Functions
- Extraction from Variants
- Assignment of Arrays to Variants
- Array Types and Functions
- Byte Array Functions
- ADO RecordSet Functions
- Debug Functions
- VB Collection Support

The following information applies only to COM Vuser scripts.

About COM Vuser Functions

Each VuGen COM function has an `Irc` prefix. VuGen records the COM API and method calls listed in this section. You can also manually program `Irc` type conversion calls. For syntax and examples of the `Irc` functions, refer to the *Online Function Reference* (**Help > Function Reference**).

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see Chapter 4, “Selecting a Script Generation Language.” The following sections describe the functions that are generated for C language type Virtual User scripts.

Creating Instances

There are several functions for creating and releasing objects, derived from the corresponding COM functions:

<code>Irc_CoCreateInstance</code>	Creates an instance of an object and returns the unknown interface.
<code>Irc_CreateInstanceEx</code>	Creates an instance of an object on a remote machine and can return multiple interfaces.
<code>Irc_CoGetClassObject</code>	Fetches the class factory for the specified class. The class factory can then be used to create multiple objects of that class.
<code>Irc_Release_Object</code>	Releases a COM object no longer in use.

IDispatch Interface Invoke Method

The following calls invoke the **IDispatch** interface using the **Invoke** method, setting different flag values in the **wflags** parameter of **Invoke**:

Irc_DispatchMethod	Invokes a method of an interface using the IDispatch:Invoke method.
Irc_DispatchMethod1	Invokes a method and gets a property of the same name using the IDispatch interface.
Irc_DispatchPropertyGet	Gets a property using the IDispatch interface.
Irc_DispatchPropertyPut	Sets a property using the IDispatch interface.
Irc_DispatchPropertyPutRef	Sets a property by reference using the IDispatch interface.

Type Assignment Functions

To supplement the functions that VuGen automatically records, you can manually program type-assignment functions into your script. The type conversion functions assign string data to the specified type. The function names are:

Irc_<Type-Name>

where *<Type-Name>* can be one of the following data types:

ascii_BSTR	ascii BSTR
bool	boolean
BSTR	BSTR
BYTE	byte
char	character variable
currency	currency
date	a date
double	double

dword	double word
float	floating point number
GUID	Returns the GUID of a named object.
hyper	hyper integer
int	integer
long	long integer
short	short integer
uint	unsigned integer
ulong	unsigned long integer
uhyper	unsigned 64-bit hyper integer
ushort	unsigned short integer

Variant Types

A variant can contain any type of information. For example, a variant may be an array of strings or a double word. A variant can also be an array of variants. VuGen can convert string data to various variant types. The functions are named:

lrc_variant_<Type-Name>

where <Type-Name> can be any of the following:

ascii BSTR	ascii BSTR variant
bool	boolean variant
BSTR	BSTR variant
BYTE	unsigned char (BYTE) variant
char	character
CoObject	an IUnknown interface pointer
currency	currency variant

date	date variant
DispObject	an IDispatch interface pointer
float	floating point number variant
int	integer variant
long	long integer variant
scode	error code variant
short	short integer variant
uint	unsigned integer variant
ulong	unsigned long variant
ushort	unsigned short variant

In addition to the variant type conversion functions, there are three functions that create new variants:

lrc_variant_empty Creates an empty variant.

lrc_variant_null Creates a null variant.

lrc_variant_variant_by_ref
Creates a new variant containing an existing variant.

Assignment from Reference to Variant

VuGen can assign variables to a reference stored inside a variant. The functions are named:

lrc_variant_<Type-Name>_by_ref

where <Type-Name> can be any of the following:

ascii BSTR	ascii BSTR variant
bool	boolean variant
BSTR	BSTR variant

BYTE	BYTE variant
char	char variant
CoObject	an IUnknown interface pointer
currency	currency variant
date	date variant
DispObject	an IDispatch interface pointer
float	floating point number variant
int	integer variant
long	long integer variant
scode	scode variant
short	short integer variant
uint	unsigned integer variant
ulong	unsigned long variant
ushort	unsigned short variant
from_variant	retrieves a variant from within a variant.

Parameterization Functions

Parameterization functions save a value of the specified type to a character string parameter. The syntaxes of parameterization functions are the following:

Irc_save_<Type-Name>

Irc_save_VARIANT_<Type-Name>

Saves a variable of the given <Type-Name> as a variant.

Irc_save_VARIANT_<Type-Name>_by_ref

Saves a variant of the given <Type-Name> as a reference within a variant.

The *value* is converted from the *<type-name>* to a character string. It is stored in a parameter. The statements are commented out by VuGen. To use them, change the name of the parameter to something meaningful and remove the statement's comments. You can then use the parameter as an input to subsequent calls. The *<type-name>* can be one of the following:

ascii_BSTR	ascii BSTR
bool	boolean
BSTR	BSTR
BYTE	byte
char	char type
currency	currency
date	a date
double	double
dword	double word
float	floating point number
hyper	hyper integer
int	integer
long	long integer
uint	unsigned integer
ulong	unsigned long integer
short	short integer
uhyper	unsigned hyper integer
ushort	unsigned short integer
VARIANT	variant

VuGen also adds a save statement for parameterization of COM scripts if you ask for correlation in a grid.

Extraction from Variants

Several functions allow extraction of data from variants:

Irc_CoObject_from_variant	Extracts a pointer to an IUnknown interface from a variant.
Irc_CoObject_by_ref_from_variant	Extracts a pointer to an IUnknown interface from a reference within a variant.
Irc_DispatchObject_from_variant	Extracts a pointer to an IDispatch interface from a variant.
Irc_DispatchObject_by_ref_from_variant	Extracts a pointer to an IDispatch interface from reference within a variant.

Assignment of Arrays to Variants

These functions convert arrays to variants:

Irc_variant_<Type-Name>Array	Assigns an array of type <Type-Name> to a variant.
Irc_variant_<Type-Name>Array_by_ref	Assigns an array of type <Type-Name> to a variant, where the array is passed by reference.

Array Types and Functions

VuGen COM supports the functions for safe arrays:

Create<n>D<Type-Name>Array	Create an array of n dimensions of the type specified in Type-Name
Destroy<Type-Name>Array	Destroy an array of the type indicated in Type-Name.

GetElementFrom<n>D<Type-Name>Array	Retrieves an element of the specified type from a SafeArray.
PutElementIn<n>D<Type-Name>Array	Stores an element in an array of the appropriate type.
Irc_Get<Type-Name>ArrayFromVariant	Extracts an array of Type-Name from a variant.
Irc_Get<Type-Name>Array_by_refFromVariant	Extracts an array of Type-Name from a pointer reference in a variant.
Fill<n>DbyteArray	Fills the last dimension of a byte array with a buffer beginning at the specified n-1 indices.

In the above functions, <Type-Name> can be one of the following data types:

Bstr	BSTR
Byte	a byte (unsigned char)
Char	a character array
CoObject	an IUnknown interface
Currency	Currency (CY)
Date	a Date variable
DispObject	an IDispatch interface
Double	double
Dword	double word
Error	an scode error
Float	floating point number
Int	integer

Long	long integer
Short	short integer
UInt	unsigned integer
ULong	unsigned long integer
UShort	unsigned short integer
Variant	a variant type

Byte Array Functions

Two sets of functions allow filling and retrieving of data from byte arrays only.

Fill<n>DByteArray	Fills the last dimension of a byte array with a buffer beginning at the specified n-1 indices.
GetBufferFrom<n>DByteArray	Gets a buffer at the specified n-1 indices from the last dimension of an n-dimensional byte array.

The **Irc_CreateVBCollection** call provides special support for a Visual Basic collection, which is a safearray of variants. VuGen treats this collection as if it were an interface. The first time it is encountered, VB creates an “interface” using **Irc_CreateVBCollection**. Thereafter, it can refer to the data at the interface address.

ADO RecordSet Functions

The following are ADO recordset functions

Irc_FetchRecordset	Moves a pointer through a recordset.
Irc_FetchRecordsetUntillEOF	Fetches records until the end of the recordset.

<code>Irc_RecordsetWrite</code>	Updates a field in an ADO recordset.
<code>Irc_RecordsetAddColumn</code>	Adds a new column to a recordset.
<code>Irc_RecordsetDeleteColumn</code>	Deletes a column from a recordset.

Debug Functions

The `Irc_print_variant` function prints the contents of a variant.

VB Collection Support

The `Irc_CreateVBCollection` function creates a Visual Basic Collection object.

27

Developing Corba-Java Vuser Scripts

VuGen allows you to record applications or applets written in Java that use Corba. You can run the recorded script or enhance it using standard Java library functions and VuGen-specific Java functions.

This chapter describes:

- ▶ Recording a Corba-Java Vuser
- ▶ Working with Corba-Java Vuser Scripts
- ▶ Recording on Windows XP and Windows 2000 Server

The following information applies to Corba-Java Vuser scripts.

About Corba-Java Vuser Scripts

Using VuGen, you can record a CORBA (Common Object Request Broker Architecture) Java application or applet. VuGen creates a pure Java script enhanced with VuGen-specific Java 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 ProTune session step.

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 (for example, *org.omg.CORBA.ORB*) must be present on the machine executing the scripts

and indicated by the *classpath* environment variable. Please refer to Chapter 23, “Programming Java Scripts” for important information about function syntax and system configuration. When recording on Windows XP and 2000 Server, follow the guidelines in “Recording on Windows XP and Windows 2000 Server,” on page 377.

The next few chapters discuss the Java recording options, run-time settings, and correlation.

Recording a Corba-Java Vuser

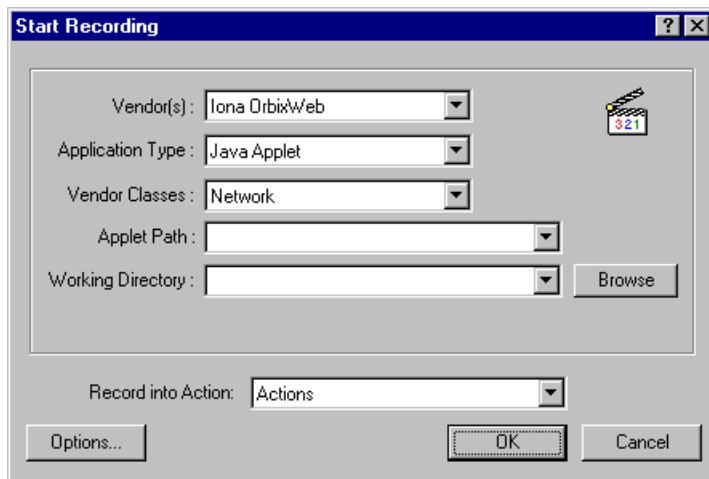
Before recording a Corba Vuser, verify that your application or applet functions properly on the recording machine.

Ensure that you have properly installed a JDK version from Sun on the machine running ProTune—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.

For more information on the required environment settings, see Chapter 23, “Programming Java Scripts.”

To begin recording:

- 1 Choose **File > New** and select Corba-Java from the Distributed Components group. The Start Recording dialog box opens.



- 2 Select a Corba vendor from the Vendor's list.
- 3 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.
 - 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`.)
- 4 In the **Vendor Classes** box, select **Network** if the Corba classes are downloaded from the network. Otherwise, when Corba classes are loaded locally, (such as JDK 1.2 and higher), only **Local** is supported.

5 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 that a Working Directory is only necessary if your application must know the location of the working directory (for example, reading property files or writing log files).

- 6** To set recording options, such as command line parameters for the JVM, click **Options**. For information about setting recording options, Chapter 13, “Setting Java Recording Options.”
- 7** In the **Record into Action** box, select the method into which you want to begin recording. The Actions class contains three methods: *vuser_init*, *action*, and *vuser_end*. The following table shows what to include into each method, and when each method is executed.

Script method	Used to emulate...	Is executed when...
<i>vuser_init</i>	a login to a server	the Vuser is initialized (loaded)
<i>action</i>	client activity	the Vuser is in “Running” status
<i>vuser_end</i>	a log off procedure	the Vuser finishes or is 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.

- 8** Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress

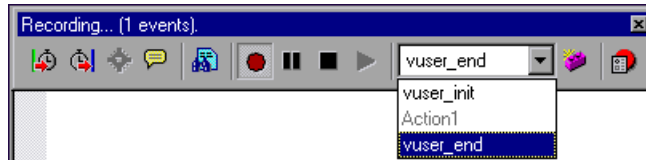
toolbar displays the names of classes as they load. This indicates that the Java recording support is active.



- 9 Perform typical actions within your application. Use the floating toolbar to switch methods during recording.



- 10 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.

- 11 Click **Stop Recording** on the Recording toolbar. The VuGen script editor displays all the recorded statements.
- 12 Click **Save** to save the script. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name.

Working with Corba-Java Vuser Scripts

Corba-specific scripts usually have a well-defined pattern. The first section contains the ORB initialization and configuration. The next section indicates the location of the Corba objects. The following section consists of the server invocations on the Corba objects. The final section includes a shutdown procedure which closes the ORB. Note that pattern is not mandatory and that each one of these sections may appear multiple times within a script.

In the following segment, the script initializes an ORB instance and performs a bind operation to obtain a Corba object. Note how VuGen imports all of the necessary classes.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.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 through out the entire recorded code appear after the *end* method and before the *Actions* class closing curly bracket.

```
// Public function: end
public int end() throws Throwable {

    if (lr.get_vuser_id() == -1)
        orb.shutdown();

    return lr.PASS;
}

// Variable section
org.omg.CORBA.ORB orb;
grid_dsi.grid grid;
}
```

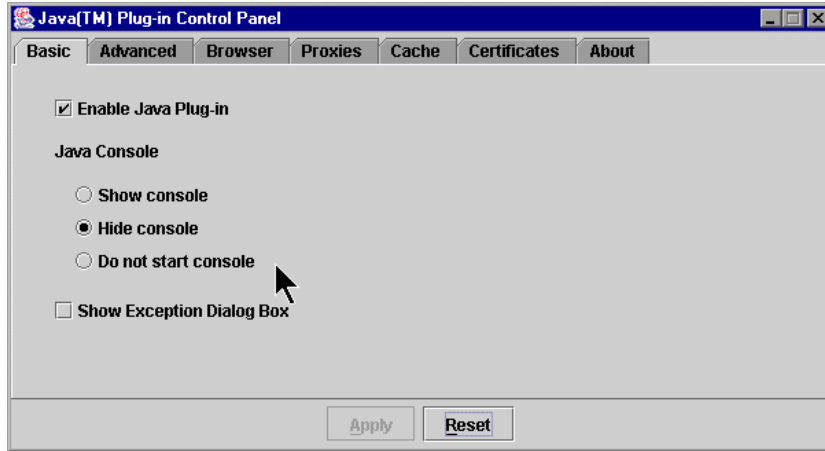
Note that the ORB shutdown statement was customized for this product. This customization prevents a single Vuser's shutdown from shutting down all other Vusers.

Recording on Windows XP and Windows 2000 Server

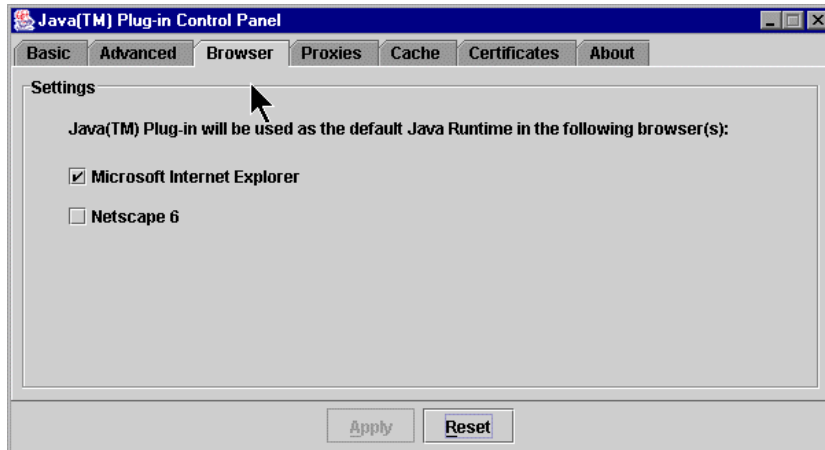
When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

To configure your machine for a Corba-Java or Rmi-Java recording:

- 1 Open the Java Plug-in from the Control Panel. Choose **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**.

28

Developing RMI-Java Vuser Scripts

VuGen allows you to record applications or applets written in Java that use RMI. You can run the recorded script or enhance it using standard Java library functions and VuGen-specific Java functions.

This chapter describes:

- Recording RMI over IIOP
- Recording an RMI Vuser
- Working with RMI Vuser Scripts

The following information applies to RMI-Java Vuser scripts.

About Developing RMI-Java Vuser Scripts

Using VuGen, you can record an RMI (Remote Method Invocation) Java application or applet. VuGen creates a pure Java script enhanced with VuGen-specific Java 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 ProTune session step.

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. Please refer to Chapter 23, “Programming Java Scripts” for important information about function syntax and system configuration.

When recording on Windows XP and 2000 Server, follow the guidelines indicated in “Recording on Windows XP and Windows 2000 Server,” on page 377.

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 an RMI Vuser

Before recording an RMI Vuser, verify that your application or applet functions properly on the recording machine.

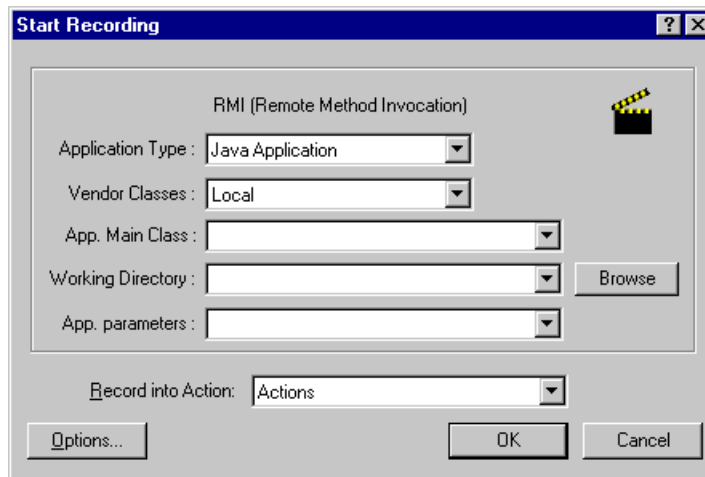
Ensure that you have properly installed a JDK version from Sun on the machine running ProTune—JRE alone is insufficient. You must complete

this installation before recording a Vuser script. Verify that the *classpath* and *path* environment variables are set according to the JDK installation instructions.

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 23, “Programming Java Scripts.”

Note that 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 ProTune.

- 1 To begin recording, choose **File > New** and select RMI- Java from the Distributed Components group. 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.

Listener mode instructs 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`.)

- 3 In the **Vendor Classes** box select **Network** or **Local**.
- 4 Specify additional parameters according for the following chart:

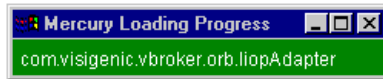
Application Type	Fields to Set
Java Applet	Applet Path, Working Directory
Java Application	App. Main Class, Working Directory, App. parameters
IExplore	IExplore Path, URL
Netscape	Netscape Path, URL
Executable/Batch	Executable/Batch, Working Directory
Listener	N/A

Note that a Working Directory is only necessary if your application must know the location of the working directory (for example, reading property files or writing log files).

- 5 To set recording options, such as command line parameters for the JVM, click **Options**. For information about setting recording options, Chapter 13, “Setting Java Recording Options.”
- 6 In the **Record into Action** box, select the method into which you want to begin recording. The Actions class contains three methods: `vuser_init`, `action`, and `vuser_end`. The following table shows what to include into each method, and when each method is executed.

Script method	Used to emulate...	Is executed when...
<code>vuser_init</code>	a login to a server	the Vuser is initialized (loaded)
<code>action</code>	client activity	the Vuser is in “Running” status
<code>vuser_end</code>	a log off procedure	the Vuser finishes or is stopped

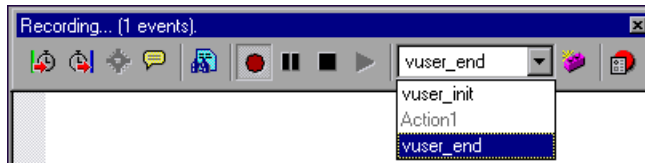
- 7 Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.



- 8 Perform typical actions within your application. Use the floating toolbar to switch methods during recording.



- 9 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.

- 10 Click **Stop Recording** on the Recording toolbar. The VuGen script editor displays all the recorded statements.
- 11 Click **Save** to save the script. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name.

Working with RMI Vuser Scripts

This section describes the elements of the Java Vuser script that are specific to RMI Vusers. 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.

In the following segment, a naming registry is located. This is followed by a lookup operation to obtain a specific Java object. We can then work with the object and perform invocations like **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 Vusers, 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 194.

Part VIII

E-Business Protocols

29

Developing FTP Vuser Scripts

VuGen allows you to emulate network activity by directly accessing an FTP server.

This chapter describes:

- Working with FTP Functions

The following information applies only to FTP Vuser scripts.

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 to an FTP server, transferring files, and logging out. To create a script, you can record an FTP session or manually enter FTP functions.

When you record an FTP session, VuGen generates functions that emulate the mail client's actions. If the communication is performed through multiple protocols such as FTP, HTTP, and a mail protocol, you can record all of them. For instructions on specifying multiple protocols, see Chapter 3, "Recording with VuGen."

To create a script for the FTP protocol, you choose the FTP protocol type in the E-Business category. To begin recording, you click the **Record** button and perform typical actions against the FTP server. For more information on creating and recording a script, see Chapter 3, "Recording with VuGen."

After you create a Virtual User script, you integrate it into a session step on either a Windows or UNIX platform. For more information on integrating

Virtual User scripts in a session step, refer to your *ProTune Console User's Guide*.

Working with FTP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 4, “Selecting a Script Generation Language.” The following section describes the functions that are generated for C language type Virtual User scripts.

FTP Vuser script functions record the File Transfer Protocol (FTP). 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.

Function Name	Description
ftp_delete[_ex]	Deletes a file from an FTP server.
ftp_dir[_ex]	Runs the dir command on the FTP server.
ftp_get[_ex]	Gets a file from an FTP server.
ftp_get_last_error	Retrieves the last error received from the FTP server.
ftp_get_last_error_id	Retrieves the ID of the last error that was received from the FTP server.
ftp_logon[_ex]	Performs a logon to an FTP server.
ftp_logout[_ex]	Performs a logout from an FTP server.
ftp_mkdir[_ex]	Creates a directory on the FTP server machine.
ftp_put[_ex]	Puts a file on an FTP server.
ftp_rendir[_ex]	Renames a directory on the FTP server machine.
ftp_rmdir[_ex]	Deletes a directory on the FTP server machine.

For the **ftp_get[_ex]**, **ftp_put[_ex]**, and **ftp_dir[_ex]** functions, you can set attributes that allow you to accurately emulate an FTP session:

PATH: The file to upload on the FTP server. (can only be used when **MSOURCE_PATH** is NOT specified)

MPATH: Specifies multiple files to upload to the FTP server.(not **ftp_dir**)

TARGET_PATH (optional): The path and filename in which to place the file on the server machine. (**ftp_put** only)

LOCAL_PATH (optional): The path and filename in which to place the file on the local machine. (**ftp_get** only)

MODE (optional): Retrieval mode ASCII or BINARY (default).

PASSIVE (optional): Sets the communication with the server to PASSIVE transmission mode.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

In the following example, the **ftp_delete** function deletes the *test.txt* file from the FTP server.

```

Actions()
{
    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();
    return 1;
}

```


30

Developing LDAP Vuser Scripts

VuGen allows you to emulate the communication with an LDAP server.

This chapter describes:

- ▶ Working with LDAP Functions
- ▶ Defining Distinguished Name Entries

The following information applies only to LDAP Vuser scripts.

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 the Defining Distinguished Name Entries section.

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 choose the LDAP protocol type in the E-Business category. To begin recording, choose **Vuser > Start**

Recording, and perform typical actions against the LDAP server. For more information on the recording procedure, see Chapter 3, “Recording with VuGen.”

After you create a Virtual User script, you integrate it into a session step on either a Windows or UNIX platform. For more information on integrating Virtual User scripts in a session step, refer to your *ProTune Console User's Guide*.

Working with LDAP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 4, “Selecting a Script Generation Language.” The following section describes the functions that are generated for C language type Virtual User scripts.

LDAP Vuser script functions emulate the LDAP protocol. Each LDAP function begins with an **mldap** prefix.

All LDAP functions come in pairs—one for global sessions and one where you can indicate a specific session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the **ex** suffix. For example, **mldap_logon** logs on to the LDAP server globally, while **mldap_logon_ex** logs on to the LDAP server for a specific session.

Function Name	Description
mldap_add	Adds an entry to the LDAP directory.
mldap_add_ex	Adds an entry to the LDAP directory for a specific session.
mldap_delete	Deletes an entry or attribute.
mldap_delete_ex	Deletes an entry or attribute for a specific session.
mldap_get_attrib_name	Gets an attribute name.
mldap_get_attrib_name_ex	Gets an attribute name a specific session.

mldap_get_attrib_value	Gets an attribute value for the current entry.
mldap_get_attrib_value_ex	Gets an attribute value for the current entry, for a specific session.
mldap_get_next_entry	Displays the next search result.
mldap_get_next_entry_ex	Displays the next search result, for the specified session.
mldap_logon	Performs a logon to an LDAP server.
mldap_logon_ex	Performs a logon to an LDAP server for a specific session.
mldap_logoff	Performs a logout from an LDAP server.
mldap_logoff_ex	Performs a logout from an LDAP server for a specific session.
mldap_modify	Modifies an entry's attribute value.
mldap_modify_ex	Modifies an entry's attribute value for a specific session.
mldap_rename	Renames an entry.
mldap_rename_ex	Renames an entry for a specific session.
mldap_search	Performs a search on an LDAP server.
mldap_search_ex	Performs a search on an 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 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

31

Creating Web Vuser Scripts

You use VuGen to develop Web Vuser scripts. VuGen creates Vuser scripts by recording your actions while you operate a client browser.

This chapter describes:

- ▶ Introducing Web Vusers
- ▶ Understanding Web Vuser Technology
- ▶ Getting Started with Web Vuser Scripts
- ▶ Recording a Web Session
- ▶ Converting Web Vuser scripts into Java

The following information applies to Web (HTML/HTTP), SOAP, and PeopleSoft8 Vuser scripts.

About Developing Web 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. The PeopleSoft8 protocol is identical to Web, with the additional capability of supporting UTF-8 character encoding.

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 session step. For details on how to integrate a Vuser script into a session step, refer to the *ProTune Console User's Guide*.

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 ensure that the response time for any customer query is less than a specified value (say 20 seconds)—even when a large number of users (say 200) access the site simultaneously. You use Vusers to emulate this case, where the Web server services the simultaneous requests for information. Each Vuser could:

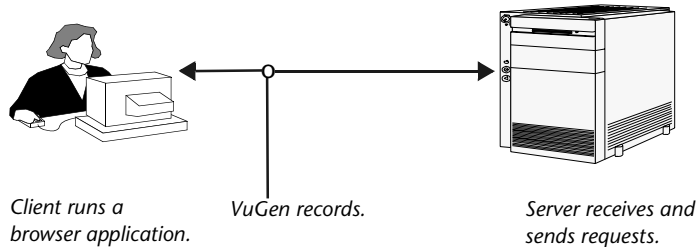
- load your home page
- navigate to the page containing the product information
- submit a query
- wait for a response from the server

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 Console, 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, either in VuGen or from the ProTune Console, 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.



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.



Select **File > New** or click the **New** button to create a new Web (HTTP/HTML) script from the e-business category, in either single or multiple protocol mode.

For details about creating a new script, see Chapter 3, “Recording with VuGen.”

2 Set the recording options.

Set the recording options. For information about setting common Internet recording options, see Chapter 34, “Setting Recording Options for Internet Protocols.” For information about Web specific recording options, see Chapter 35, “Setting Recording Options for Web Vusers.”

3 Record a browser session.

Record your actions while you navigate your Web site.

For details about creating a new script, see Chapter 3, “Recording with VuGen.”

4 Enhance the recorded Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, checks, and service steps.

For details, see Chapter 38, “Verifying Web Pages Under Load,” Chapter 39, “Modifying Web and Wireless Vuser Scripts,” and Chapter 40, “Configuring Correlation Rules for Web Vuser Scripts.”

5 Define parameters (optional).

Define parameters for the fixed values recorded into your script. By substituting fixed values with parameters, you can repeat the same Vuser action many times using different values.

For details, see Chapter 7, “Defining Parameters.”

6 Configure the run-time settings.

The run-time settings control Vuser behavior during script execution. These settings include general run-time settings (iteration, log, think time, and general information), and Web-related settings (proxy, network, and HTTP details).

For details, see Chapter 9, “Configuring Run-Time Settings.”

7 Perform correlation.

Scan your Vuser script for correlations and use one of VuGen’s mechanisms to implement them. For more information, see Chapter 40, “Configuring

Correlation Rules for Web Vuser Scripts” or Chapter 41, “Correlating Vuser Scripts After Recording.”

8 Run and debug the Vuser script using VuGen.

Run the Vuser script from VuGen to verify that it runs correctly. For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode” and Chapter 43, “Using Reports to Debug Vuser Scripts.”

After you create a Vuser script, you integrate it into a ProTune session step. For more information on integrating Vuser scripts into a session step, refer to the *ProTune Console User’s Guide*.

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 login 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.

During recording, VuGen places functions and their resources in *Concurrent Groups*. A concurrent group represents links and resources that are loaded on a page at the same time. For example, it is common for browsers to begin loading a second or third image while the first image is being loaded. Statements in a concurrent group are enclosed with **web_concurrent_start** and **web_concurrent_end** functions. During replay, when ProTune encounters a **web_concurrent_start** statement, it registers the functions, but does not execute them until the group is closed with **web_concurrent_end**. For more information about the concurrent group functions, see the *Online Function Reference* (**Help > Function Reference**).

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 the <PROTUNE>/dat directory.
- 6** Type the following command:

```
<PROTUNE>\bin\sed -f web_to_java.sed filename > outputfilename
```

where *filename* is the full path and file name 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.

32

Using Web Vuser Functions

You use VuGen to develop Web Vuser scripts. VuGen creates Vuser scripts by recording your actions while you operate a client browser.

This chapter describes:

- ▶ Adding and Editing Functions
- ▶ Web Function List
- ▶ Viewing Scripts in the Tree View
- ▶ Viewing Vuser Scripts in Script View

The following information applies to Web (HTML/HTTP), SOAP, Wireless, and PeopleSoft8 Vuser scripts.

About Web Vuser Functions

The functions developed to emulate Internet communication between a browser or toolkit 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.

For detailed information and examples of the Internet Protocol functions (Web, Wireless, etc.) refer to the *Online Function Reference* (**Help > Function Reference**).

VuGen can display a Web or Wireless Vuser script in two ways:

- As an icon-based representation of the Vuser script. This is the default view, and is known as the *tree view* (not available for WAP Vusers). For more information, see “Viewing Scripts in the Tree View,” on page 412.
- As a text-based representation of the Vuser script. This is known as the *script view*. For more information, see “Viewing Vuser Scripts in Script View,” on page 414.

Adding and Editing Functions

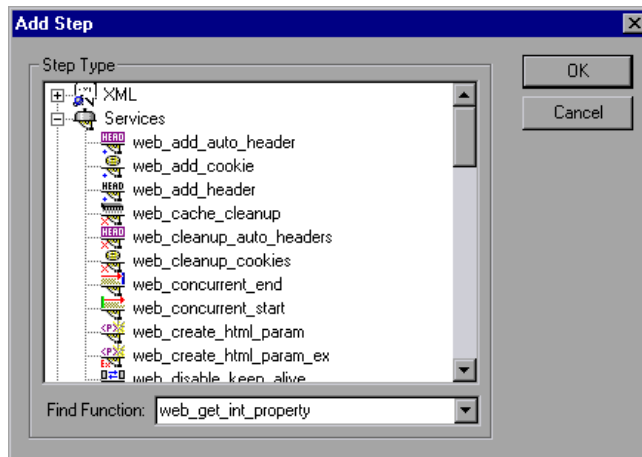
Many of the Web Vuser functions are recorded during the browser or toolkit session.

You can manually add general Vuser functions such as transactions, rendezvous, comments, and log functions during recording. For more information, see Chapter 6, “Enhancing Vuser Scripts.”

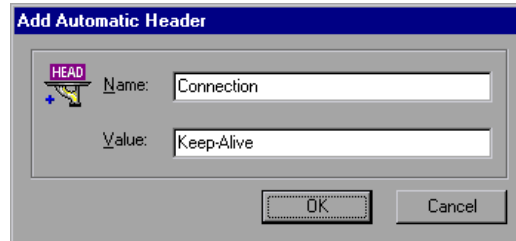
This section describes how to add and edit Web Vuser functions during and after recording in both Tree view and Script view.

To add a new function to an existing Vuser script:

- 1 Choose **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. The Properties dialog box for each function 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**.

Web Function List







The Web Vuser functions that represent communication over the Internet, begin with the **web** prefix. The Web functions are categorized as follows:

- Action Functions
- Authentication Functions
- Cache Function
- Check Functions
- Connection Definition Functions

- ▶ Concurrent Group Functions
- ▶ Cookie Functions
- ▶ Correlation Functions
- ▶ Filter Functions
- ▶ Header Functions
- ▶ Proxy Server Functions
- ▶ Miscellaneous Functions

Action Functions

When you record a Web Vuser script, VuGen generates the following action functions, and inserts them into the script:

	web_custom_request	Allows you to create a custom HTTP request with any method supported by HTTP.
	web_image	Emulates a mouse click on the defined image.
	web_link	Emulates a mouse click on the defined text link.
	web_submit_data	Performs an "unconditional" or "contextless" form submission.
	web_submit_form	Emulates the submission of a form.
	web_url	Loads the URL specified by the "URL" attribute.

Authentication Functions



`web_set_certificate`

Causes a Vuser to use a specific certificate that is listed in the Internet Explorer registry.



`web_set_certificate_ex`

Specifies location and format information of a certificate and key file.



`web_set_user`

Specifies a login string and password for a Web server, for user-authenticated areas in the Web server.

Cache Function



`web_cache_cleanup`

Clears the contents of the cache simulator.

Check Functions



`web_find`

Searches inside an HTML page for a specified text string.



`web_global_verification`

Searches for a text string in all subsequent HTTP requests.



`web_image_check`

Verifies the presence of a specified image inside an HTML page.



`web_reg_find`

Registers a search for a text string in an HTML source or raw buffer, in the subsequent HTTP request.

Connection Definition Functions



web_disable_keep_alive Disables keep-alive HTTP connections.



web_enable_keep_alive Enables keep-alive HTTP connections.



web_set_connections_limit Sets the maximum number of simultaneous connections that a Vuser can open when running a script.

Concurrent Group Functions



web_concurrent_end Marks the end of a concurrent group.



web_concurrent_start Marks the beginning of a concurrent group.

Cookie Functions



web_add_cookie Adds a new cookie or modifies an existing one.







web_cleanup_cookies Removes all the cookies that are currently stored by the Vuser.






web_remove_cookie Removes the specified cookie.




Correlation Functions

	<code>web_create_html_param</code>	Saves dynamic information on an HTML page to a parameter.(LR 6.5 and below)
	<code>web_create_html_param_ex</code>	Creates a parameter based on the dynamic information contained in an HTML page - uses embedded boundaries. (LR 6.5 and below)
	<code>web_reg_save_param</code>	Creates a parameter based on the dynamic information contained in an HTML page - does not use embedded boundaries.
	<code>web_set_max_html_param_len</code>	Sets the maximum length of retrieved dynamic HTML information.

Filter Functions

	<code>web_add_filter</code>	Sets criteria to includes or exclude URL's when downloading.
	<code>web_add_auto_filter</code>	Sets criteria to includes or exclude URL's when downloading
	<code>web_remove_auto_filter</code>	Disables filtering of download content.

Header Functions

	<code>web_add_auto_header</code>	Adds a customized header to all subsequent HTTP requests.
	<code>web_add_header</code>	Adds a customized header to the next HTTP request.
	<code>web_cleanup_auto_headers</code>	Stops adding customized headers to subsequent HTTP requests.



web_remove_auto_header

Stops adding a specific header to subsequent HTTP requests.



web_revert_auto_header

Stops adding a specific header to subsequent HTTP requests, but generates implicit headers.



web_save_header

Saves request and response headers to a variable.

Proxy Server Functions



web_set_proxy

Specifies that all subsequent HTTP requests be directed to the specified proxy server.



web_set_proxy_bypass

Specifies the list of servers that Vusers access directly, that is, not via the specified proxy server.



web_set_proxy_bypass_local

Specifies whether or not Vusers should bypass the proxy for local (intranet) addresses.



web_set_secure_proxy

Specifies that all subsequent HTTPS requests be directed to the server.

Replay Functions



web_set_max_retries






Sets the maximum number of retries for an Action step.



web_set_timeout





Specifies the maximum amount of time that a Vuser waits to execute a specified task.

Miscellaneous Functions

	<code>web_convert_param</code>	Converts an HTML parameter to a URL or plain text.
	<code>web_get_int_property</code>	Returns specific information about the previous HTTP request.
	<code>web_report_data_point</code>	Specifies a data point and adds it to test results.
	<code>web_set_option</code>	Sets a Web option in the area of encoding, redirection, and downloading of non-HTML resources.
	<code>web_set_sockets_option</code>	Sets an option for sockets.

Control Type Functions

In addition to Web Vuser functions, the following control functions may also appear in your Vuser script:

	<code>lr_start_transaction</code>	Marks the beginning of a transaction for performance tuning.
	<code>lr_end_transaction</code>	Marks the end of a transaction for performance tuning.
	<code>lr_rendevous</code>	Sets a rendezvous point in the Vuser script.
	<code>lr_think_time</code>	Pauses execution between commands in a Vuser script.

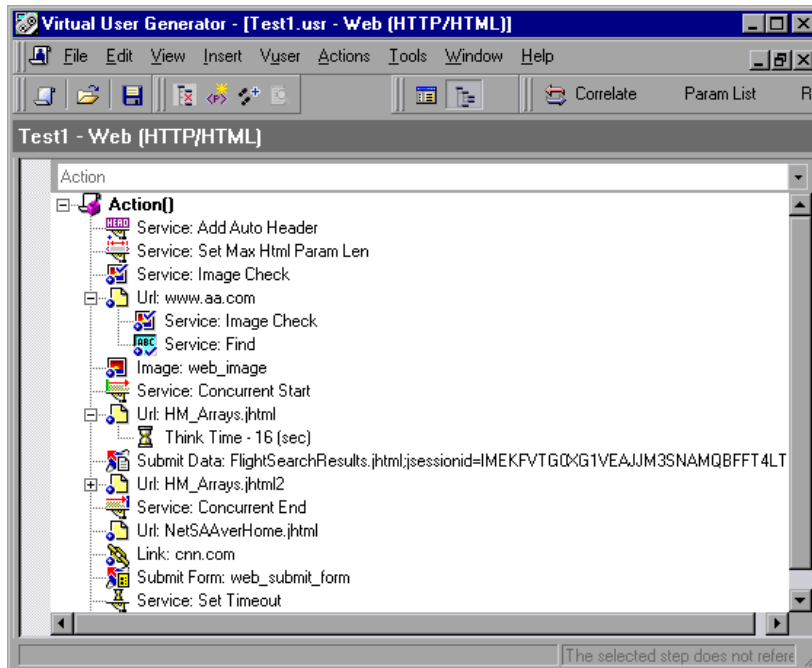
For more information on adding general Vuser functions to scripts, see Chapter 6, “Enhancing Vuser Scripts.”

Viewing Scripts in the Tree View

The Tree view displays Vuser scripts in an icon-based view. Each Vuser function is represented by an icon. This view is not available for WAP Vusers.

To display the tree view of a Web Vuser script:

- From the VuGen main menu, select **View > Tree View**, or click the **View script as tree** button. The Vuser script is displayed in the icon-based tree view. If you are already in the tree view, the menu item is disabled.



The tree view of a Vuser script is composed of icons representing an action of the Vuser or a step in the Vuser script. The step type is indicated by the string preceding the step name. For example:

- Url: web_url
- Image: GIF.GIF
- Submit Form: web_submit_form
- Submit Data: web_submit_data
- Custom Request: web_custom_request
- Link: web_link

The following step types are supported in VuGen:

Icon Type	Description
Service	A <i>Service</i> 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 icon is added to the Vuser script when you type in a URL 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 icon when you click a hypertext link while recording. Each Link icon represents a web_link function in the Vuser script. The default label of the icon is the text string of the hypertext link.
Image	VuGen adds an Image icon to the Vuser script when you click a hypergraphic link while recording. Each Image icon 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.
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.
Custom Request	VuGen adds a Custom Request step to a Vuser script when you record an action that VuGen can not recognize as any of the standard actions (i.e., URL, link, image, or form submission). This is applicable to non-standard HTTP applications.

Note: The Link, Image and Form Submission steps are only recorded when you select the option to record in HTML-based mode. For more information, refer to Chapter 35, “Setting Recording Options for Web Vusers.”

Viewing Vuser Scripts in Script View

To view and edit the text-based representation of a Web Vuser script, you select the script view.

To display the script view of a Web Vuser script:

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



```

IP_think_time(16);

web_concurrent_end(NULL);
web_url("NetSAAverHome.jhtml",
        "URL=http://www.aa.com/apps/netSAAver/NetSAAve
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.aa.com/apps/reservations/F
        "Snapshot=t6.inf",
        "Mode=HTML",
        LAST);
web_link("cnn.com",
        LAST);
web_submit_form("web_submit_form",
        ITEMDATA,
        LAST);
web_set_timeout(CONNECT,
        "120");

web_url("HM_Arrays.jhtml3",

```

In the script view, you can see the functions that were generated by your browser application, and you can make changes to the script as required.

Note: If you make changes to a Vuser script while in the script view, VuGen makes the corresponding changes in the tree view of the Vuser script. If VuGen is unable to comprehend the text-based changes that were made, VuGen will be unable to convert the script view into tree view.

33

Recording Web/WinSock and SOAP Vuser Scripts

VuGen lets you create a Web/WinSock dual protocol Vuser script that emulates applications accessing the Web and Windows Sockets. A popular use for this protocol is the Palm HotSync process.

This chapter describes:

- ▶ Getting Started with Web/WinSock Vuser Scripts
- ▶ Setting Browser and Proxy Recording Options
- ▶ Setting Web Trapping Recording Options
- ▶ Recording a Web/WinSock Session
- ▶ Recording Palm Applications

The following information only applies to the Web/Windows Sockets Dual Protocol, SOAP, and Palm Vuser scripts.

About Recording Web/WinSock Vuser Scripts

VuGen's Web/WinSock dual protocol type, lets you successfully record non-HTML Web applications. VuGen records these applications using both Web and Windows Sockets protocol functions and creates a script that emulates access to Web pages and socket activity. A common application for this protocol is the recording of a HotSync process of a handheld organizers using the Palm OS protocol. VuGen records the transfer of data and generates the relevant functions. Note that wireless data transfers for the Palm, are not recorded.

When you run the dual protocol script, the Vuser emulates activity between a Web browser, the non-HTML application, and the Web Server. The dual protocol capabilities allow you to record only once for both the Web and WinSock protocols, thus avoiding any duplicate calls. VuGen synchronizes the recordings of the two protocols and creates a single script containing both Web and WinSock Vuser functions.

Preferably, you should record a Web and WinSocket session using a multi-protocol script, specifying the Web and WinSock protocols. The multi-protocol mode, however, does not support UDP sockets, so if you need to record UDP sockets, use the Dual Web/WinSock Vuser discussed in this chapter.

The WinSock functions represent in low level code the socket activity during the recorded session. Each WinSock function begins with an **lrs** prefix and relates to the sockets, data buffers, and environment. You can also view the actual data that was sent and received during the session by selecting *data.ws* in VuGen's left pane. Note that recording of UDP types sockets is not supported in this mode.

The Web functions begins with a **web** prefix. These functions relate to standard Web actions such as going to a URL (**web_url**), submitting data (**web_submit_data**), and adding cookies (**web_add_cookie**).

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

After you record the dual protocol script, you can edit it by modifying the text of the script in the script view. Note that tree view and Snapshot window, which are available for standard Web Vuser scripts, are not supported for Web/WinSock scripts.

You correlate values in your Web/WinSock Vuser script just as you would in a single protocol script. You must, however, correlate Web functions according to the Web correlation procedure, and the WinSock functions according to their procedure. For information on correlating Web functions, see Chapter 40, "Configuring Correlation Rules for Web Vuser Scripts." For details on correlating WinSock functions, see Chapter 8, "Correlating Statements."

Getting Started with Web/WinSock Vuser Scripts

This section provides an overview of the process of developing a dual protocol Web/WinSock Vuser script using VuGen.

To develop a Web/WinSock Vuser script:

1 Record the basic script using VuGen.

Invoke VuGen and create a new Vuser script. Specify Web/Winsocket Dual Protocol as the type of Vuser. Choose an application to record and set the Web and WinSock recording options. Record typical operations on your application.

For details, see “Setting Browser and Proxy Recording Options,” on page 420

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed values recorded into your script. By substituting fixed values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, “Correlating Statements” or Chapter 40, “Configuring Correlation Rules for Web Vuser Scripts.”

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 9, “Configuring Run-Time Settings.”

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Vuser script, you integrate it into a ProTune session step. For more information, refer to the *ProTune Console User's Guide*.

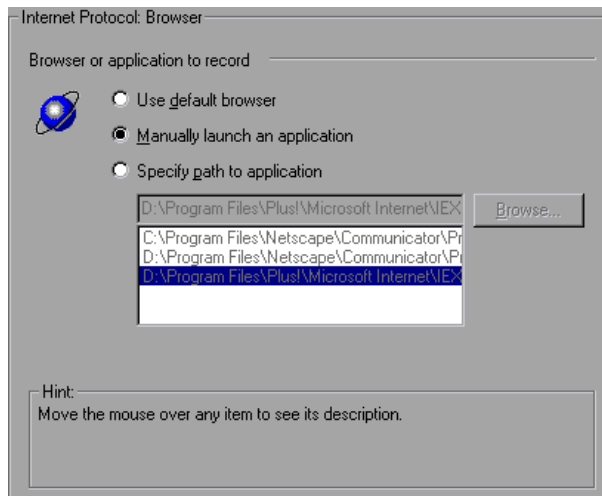
Setting Browser and Proxy Recording Options

Before recording a script, you set the Web and WinSock recording options. You set the Web recording options in the following areas: Browser, Proxy, Recording Information, and Correlation. You set the WinSock recording options to exclude sockets, set a think time threshold value and specify a translation table. This section describes the Browser and Proxy recording options. For information on other Internet protocol recording options, see Chapter 34, “Setting Recording Options for Internet Protocols.” For information on WinSock recording options, see Chapter 20, “Developing WinSock Vuser Scripts.”

To open the recording options, choose **Tools > Recording Options** or click the **Options** button in the Start Recording dialog box.

Setting the Browser Recording Options

The Browser recording options let you specify which browser VuGen uses when you record a Vuser script.



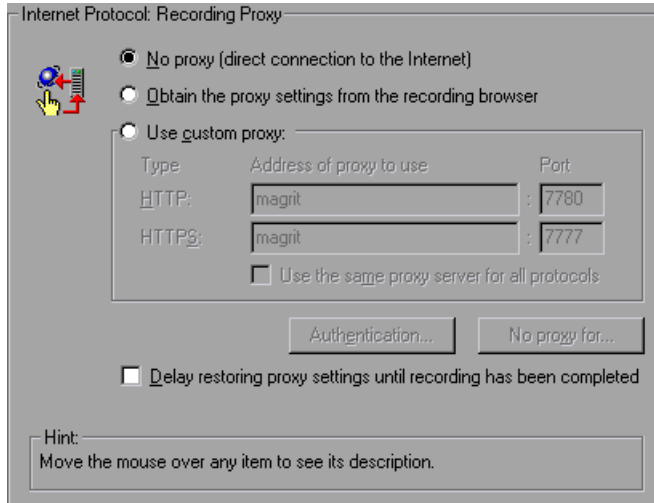
Select one of the following three options on the **Browser** tab. Note that these options are only relevant when Web trapping is disabled (see “Setting Web Trapping Recording Options” on page 423). If you enable Web trapping, the application in the **Program to Record** field is always launched.

- **Use default browser**, to instruct VuGen to use the default Web browser on the recording computer. The application in the **Program to Record** field of the Start Recording dialog box is ignored. You must, however, enter a value into this field, even though it is not used. Use this option to record Active-X applications or Java templates.
- **Manually launch an application**, to instruct VuGen not to automatically launch an application (in this case a browser) when you start recording. You specify the browser’s path in the **Program to Record** field of the Start Recording dialog box, and VuGen launches it when it begins recording, and prompts you to modify the proxy settings (see page 421). Use this option for a standalone application or for an application that invokes a browser.
- **Specify path to application**, to instruct VuGen to automatically start a specific application. Select an application and its path from the list, or click the **Browse** button to locate the desired one. The application in the **Program to Record** field of the Start Recording dialog box is ignored. You must, however, enter a value into this field, even though it is not used. Use this option to use a non-browser application or a browser other than the default one.

Specifying the Recording Proxy Settings

If you set the recording option to manually launch the browser (see previous section), and you do not enable Web Trapping, you may need to adjust the proxy setting. Since you are not automatically invoking a browser, you cannot instruct VuGen to obtain the proxy settings from the recording browser.

Instead, select the **Always use direct connection to the Internet** option.



After you begin recording, VuGen issues a message indicating that you should change your browser's proxy settings and what those settings should be.



If you click **OK** without modifying your browser's settings, VuGen will only record the application and not the browser actions. To set the proxy settings, abort the recording and set the browser settings.

To modify the proxy settings:

- For Netscape, choose **Edit > Preferences > Advanced > Proxies > Manual Proxy Configuration** and enter *localhost* (lower case) for the host name and the port number provided in the above dialog box.
- For Internet Explorer, choose **Tools > Internet Options > Connections > LAN Settings** and select **Use a Proxy Server**. Enter *localhost* (lower case) for the host name and the port number provided in the above dialog box.

For information on additional Web recording options, see Chapter 35, “Setting Recording Options for Web Vusers.” For information on WinSock recording options, see Chapter 20, “Developing WinSock Vuser Scripts.”

Setting Web Trapping Recording Options

When VuGen records a script for a Web/WinSock Vuser, it modifies your browser’s proxy settings. VuGen directs all HTTP and HTTPS requests through the reconfigured proxy ports. After directing Web requests through the proxy ports, it directs them to the ports specified in the **Recording Proxy** tab. All requests that are not sent or received via the specified proxy ports, are recorded as WinSock functions and not HTTP Web requests. After recording, VuGen restores all of the original proxy settings.

Certain applications issue Web events, but do not support proxy configuration, such as certain Java applets. VuGen cannot set the required internal proxy settings for these applications. As a result, these applications are not recorded as Web events and the events are recorded as WinSock requests, making them less readable and less intuitive. For information on how to record the applications and their startups, see “Setting the Browser Recording Options” on page 420.

The Web Trapping settings allow you to trap or save an event that would normally be recorded as a WinSock function, as a Web function. When you enable the trapping option, VuGen waits for events at a specific port, marks them as Web events, and generates the appropriate Web functions. This results in a more readable and intuitive script.

You need to specify the port at which VuGen should listen for Web events. All communications on that port are handled as Web events, represented by

Web Vuser functions. You can use the default ports-80 for HTTP and 443 for HTTPS, or you can specify any IP:port combination. VuGen supports wildcard combinations, to include all ports on a particular host.

For example 207.232.15.30:* indicates all ports on the host machine 207.232.15.30. The entry 207.232.*.*:80 indicates standard port 80 on all machines in the domain of 207.232. Note that you cannot mix digits and wildcards within the sections of an IP address. For example, 207.2*.32.9 is an invalid entry.

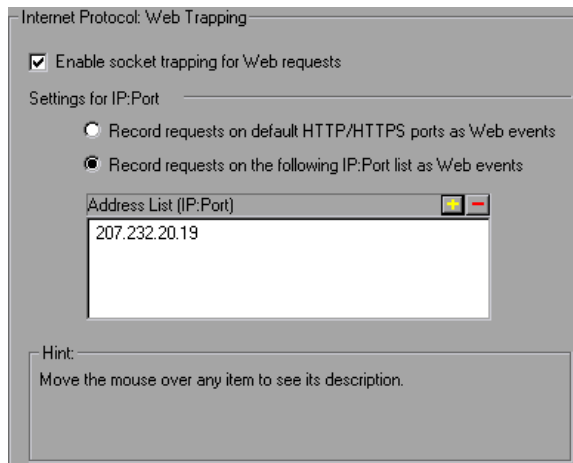
To determine whether or not to enable Web trapping, first perform a recording session. View the data file, *data.ws*. If you see HTTP or HTTPS data that was recorded as WinSock buffer data, this may indicate that the request was made over a different port. In this instance, you should enable Web trapping to allow VuGen to generate Web functions for those requests.

This option is especially useful when you manually launch the application to record, instead of recording through a browser. For information about manually launching an application, see “Setting the Browser Recording Options,” on page 420.

Note that when you enable Web trapping, all Windows Sockets communication on the specified ports is ignored.

To set the Web Trapping recording options:

- 1 Choose **Tools > Recording Options** and select the Web Trapping node.




- 2 To enable trapping for Web events, select the **Enable socket trapping for Web requests** check box.
- 3 To trap Web events on the default ports, choose **Record requests to default HTTP/HTTPS ports as Web events**.
- 4 To trap Web events on ports other than the default, choose **Record requests to the following IP:Port. list as Web Events**. Click the “+” to add a new IP:port entry to the list. Click “-” to remove an existing entry. You can use wildcards as described in the previous section.
- 5 Click OK to save the settings and close the dialog box.

Recording a Web/WinSock Session

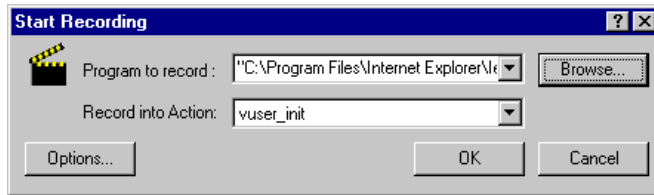
You record a dual protocol session in a similar way as you would record standalone Web and Windows Sockets Vusers. When you record a dual protocol session, VuGen monitors all the actions that you perform within your Web browser or application, and generates the appropriate Web or WinSocket function.

Each script that you create contains three sections: *vuser_init*, *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 login and logoff procedures, which are not repeated when you run a script with multiple iterations. You should therefore record in the *Actions* section, so that the complete browser session is repeated for each iteration.

To record a Web/WinSock session:

- 1 Open the recording browser, and set the home page to the URL you want to record.
- 2 Select **Start > Programs > ProTune > Virtual User Generator**. The VuGen main window opens.
- 3  Create a new Web/WinSock script. Choose **File > New** or click the **New** button.

- 4 Select **Web/Winsocket Dual Protocol** from the **E-Business** folder, and click **OK**. VuGen opens a skeleton Vuser script and displays the Start Recording dialog box.





- 5 Click **Options** to set the recording options for the socket, browser, proxy, or other advanced settings. If you are recording with a browser, specify a browser. If you are recording a non-browser application (such as streaming data), set the Browser Recording Option to **manually launch a browser**. For manual launching, set the proxy option to **Always use direct connection to the Internet Proxy** and modify your browser's proxy setting to *localhost*. Refer to "Setting the Browser Recording Options," on page 420 for additional information on setting these recording options.
- 6 Click **Browse** to select the program to record. Note that this entry is only used when you specify **manually launch a browser** in the recording options (**Browser** node). Specify the path and name of the non-browser application in the **Program to Record** box. If you are recording with a browser, this entry is ignored; you must, however, enter a value into this box.
- 7 From the **Record into Action** list, select the section into which you want to begin recording.
- 8 Click **OK** to launch the application and start recording. The floating recording toolbar appears.



Note: When recording a Web Vuser script, you can only run a single instance of Netscape Navigator. Therefore, if Netscape Navigator is running before you begin recording, VuGen prompts you to close the browser. This enables VuGen to open the Netscape browser itself.

- 9 Perform the desired business process. Each link you click adds a **web_url** function to the script. Each form you submit adds a **web_submit_form** function to the Vuser script. Non-browser application actions are recorded as socket data.

During recording, you can use the VuGen floating toolbar to insert transactions, rendezvous points, and instant text checks. For more details, see below. For details on inserting text or image checks, see Chapter 38, “Verifying Web Pages Under Load.”

-  **10** After performing all the required user processes, click the **Stop recording** button on the floating recording toolbar. VuGen restores the VuGen main window.
-  **11** Choose **File > Save** or click the **Save** button to save the Vuser script. Specify a file name and location in the Save Test dialog box, and click **Save**.

After recording, you can edit the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script. For details, see Chapter 6, “Enhancing Vuser Scripts.”

After modifying a script, you can revert back to the originally recorded version of the script, using the *Regenerate Vuser* utility. This utility only regenerates the WinSock statements; it does not affect the Web statements. For more information, see “Regenerating a Vuser Script,” on page 40.

Recording Palm Applications

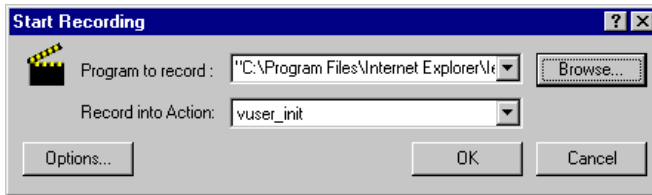
Palm-based applications offer two ways to communicate with a remote server: cradle and wireless. Palm application docked on a cradle communicate directly with their servers over the Internet through the HotSync service. VuGen allows you to capture all traffic channeled through

Palm's HotSync service. Since many applications use HTTP as a transport layer to communicate to their server, the script generated is web-like, and inherits the same syntax and functionality as Web. In rare occasions, the traffic is channeled over a proprietary protocol. This proprietary traffic will also be recorded and represented as WinSock functions in the script.

To record a Palm application:



- 1 Create a new script. Choose **File > New** or click the **New** button.
- 2 Select **Palm** from the **E-Business** folder, and click **OK**. VuGen opens a skeleton Vuser script and displays the Start Recording dialog box.



- 3 Specify HotSync.Exe as the application to record, and click **OK**.
Make sure that HotSync.Exe is not already running prior to launching it from VuGen.
- 4 Set the Palm Pilot on the Cradle, and interact with your applications.
Note that you may need to press the **HotSync** button on your Palm Pilot to initiate the communication between the Palm and the server.
- 5 After performing all the required user processes, click the **Stop recording** button on the floating recording toolbar. VuGen restores the VuGen main window.
- 6 Choose **File > Save** or click the **Save** button to save the Vuser script. Specify a file name and location in the Save Test dialog box, and click **Save**.

After recording, you can edit the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script. For details, see Chapter 6, "Enhancing Vuser Scripts."

The script is represented as a combination of Web and WinSock protocols. All Palm traffic that was carried over HTTP is represented in **web_url**

statements and **web_submit_data** requests. Proprietary protocols are represented by calls to WinSock functions.

34

Setting Recording Options for Internet Protocols

For protocols that work over the Internet, you can customize the Internet related recording options.

This chapter describes:

- ▶ Working with Proxy Settings
- ▶ Setting Advanced Recording Options
- ▶ Setting a Recording Scheme

The following information only applies to Web, Wireless, and Oracle NCA protocols.


About Setting Recording Options for Internet Protocols

VuGen creates Vuser scripts that emulate a true Internet environment.

Before recording, you can configure VuGen's recording options relating to the proxy and script generation preferences.

You can also set protocol specific recording options for Web Vuser scripts. For more information, see the Recording Options chapter for your protocol.

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

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

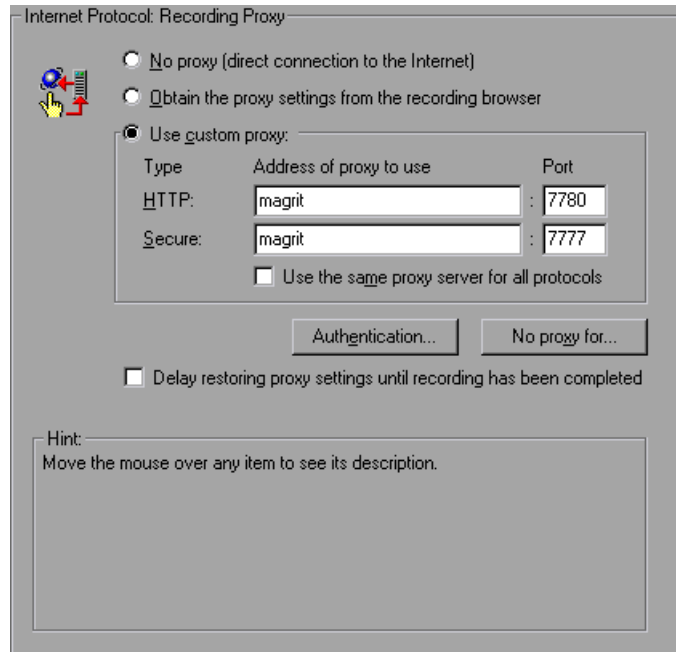
Working with Proxy Settings

A proxy server is a server that resides between a client (such as a Web browser) and a Web server. It intercepts all requests sent to the server and attempts to fulfill these requests. Proxy servers are used for two primary reasons—to improve performance and filter requests. To improve performance, it stores Web pages accessed by one user and makes them available to another user without accessing the server a second time. A proxy server also lets an administrator filter the content that can be viewed in browsers.

To use a proxy server, you specify its name or IP address in your browser's preferences. In typical cases, Internet Service Providers recommend that their users connect through a proxy server, and companies require their employees to access the Internet through a proxy server.

By default, VuGen uses the proxy settings from the recording browser. VuGen also lets you customize the proxy settings for the recording session. If you know in advance that your users access the Internet directly without going through a proxy server, or that users will be using a specific proxy server, other than your browser's default, you can customize the proxy settings. To customize the settings, select the **Internet Protocol:Recording**

Proxy node in the Recording Options tree and modify the recording proxy settings.

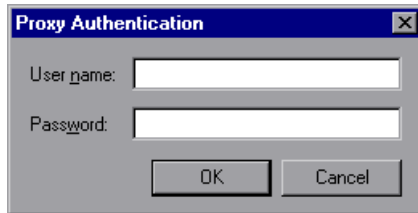


You can choose one of the following proxy options:

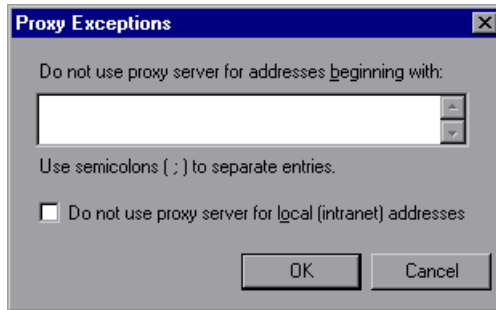
- **No proxy (direct connection to the Internet):** Always use a direct connection to the Internet. This means that a direct connection is made without using a proxy server. This usually corresponds with the Internet Explorer setting of Automatically Detect Settings.
- **Obtain the proxy settings from the recording browser:** Use the proxy settings from the recording browser. This is the default option. This option is not available for Web/WinSock Vusers.
- **Use custom proxy.** Use the specified proxy server during recording. You can specify a proxy server for all non-secure HTTP sites and another proxy server for all secure (HTTPS) sites. This section is only enabled when the two above options are cleared.

If the HTTP and HTTPS proxy servers are the same, specify only the HTTP address and port, and select the **Use the same proxy server for all protocols** option.

Some proxy servers require authentication with a user name and password. If you are recording a session through a proxy that requires authentication, click the **Authentication** button and supply the relevant **User name** and **Password** in the Proxy Authentication dialog box.



To specify host names or IP addresses that you want VuGen to access directly, that is, without using a proxy server, click the **No proxy for** button. The Proxy Exceptions dialog box opens.



Type the addresses that you want VuGen to access directly. Separate each address with a semicolon.

To specify that VuGen should not use the proxy server when it accesses local (intranet) addresses, select the **Do not use proxy server for local (intranet) addresses** option.

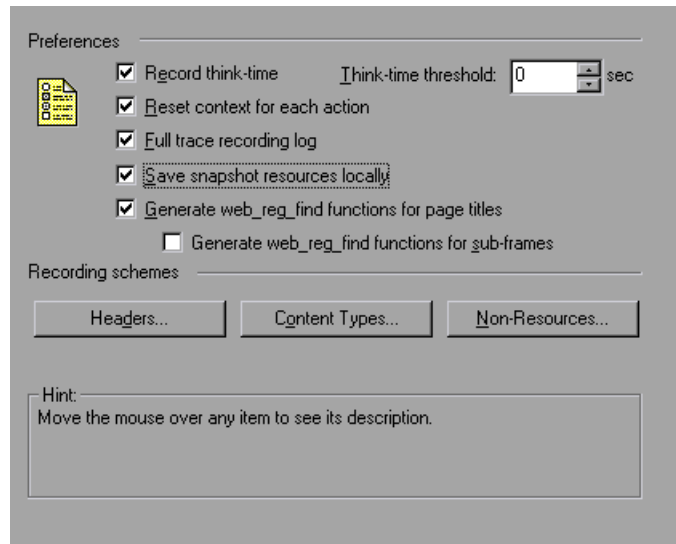
Restoring Proxy Settings

If you specify proxy setting for recording that are different from the machine's regular browser settings, VuGen restores the original browser settings. By default, VuGen restores the original proxy settings immediately after the launched browser reads them. To restore the original proxy settings only after you stop recording, select the **Delay restoring proxy settings until recording has completed** check box. This option only applies to Internet Explorer.

Optimally, you should restore the proxy settings immediately to insure the security of your machine. The option to restore the settings after recording is less secure, but is required when the proxy settings might be read later. This occurs, for example, when you are recording HTTP actions on applets, ActiveX controls, and multi-window applications.

Setting Advanced Recording Options

Use the **Internet Protocol:Advanced** settings to allow the recording of custom headers, apply a content type filter, specify non-resources, and set other code generation preferences.



Recording Think Time (Web and Wireless)

Think time emulates the time that a real user waits between actions. To record user think time, select **Record think time**. You can specify the minimum amount of time that a user waits that should be recorded as think time by defining a **Think time threshold**. For example, you can set the think time threshold to 5, so if a user waits for less than five seconds, think time is not recorded.

Use the Think Time run-time settings to influence how the Vuser uses the recorded think time when you run the script.

Reset Context for Each Action (Web and Oracle NCA)

You can instruct VuGen to reset all HTTP contexts between actions. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. This option resets the HTML context, so that a contextless function is always recorded in the beginning of the action. It also clears the cache and resets the user-names and passwords. (enabled by default)

Creating a Trace Log (Web only)

Select **Full trace recording log** to create a trace log during recording. This log is used internally by Mercury Interactive Customer Support. (disabled by default)

Saving Snapshot Information Locally

Select **Save snapshot resources locally** to instruct VuGen to save a local copy of the snapshot resources during record and replay. This feature lets VuGen create snapshots more accurately and display them quicker.

Generate web_reg_find functions for page titles (Web and Oracle NCA)

Select **Generate web_reg_find functions for page titles** to enable the generation of **web_reg_find** functions for all HTML page titles. VuGen adds the string from the page's title tag and uses it as an argument for **web_reg_find**. Select **Generate web_reg_find functions for sub-frames** to enable the generation of **web_reg_find** functions for page titles in all sub-frames of the recorded page.

Setting a Recording Scheme

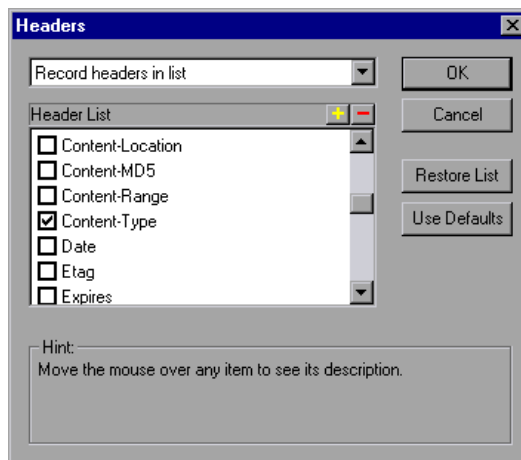
You can further customize the recording by specifying a recording scheme in the following areas:

- Recording Custom Headers
- Filtering Content Type
- Specifying Non-Resource Content Types

Recording Custom Headers

Web Users automatically send several standard HTTP headers with every HTTP request submitted to the server. Click **Headers** to instruct VuGen to record additional HTTP headers. You can work in three modes: **Do not Record Headers**, **Record Headers in list**, or **Record Headers not in list**. When you work in the first mode, VuGen does not record any headers. In the second mode, VuGen only records the checked custom headers. If you specify **Record headers not in list**, VuGen records all custom headers except for those that are checked and other risky headers.

The following standard headers are known as *risky* headers: Authorization, Connection, Content-Length, Cookie, Host, If-Modified-Since, Proxy-Authenticate, Proxy-Authorization, Proxy-Connection, Referer, and WWW-Authenticate. They are not recorded unless selected in the Header list. The default option is **Do not Record Headers**.



In the **Record Headers in List** mode, VuGen inserts a `web_add_auto_header` function into your script for each of the checked headers that it detects. This mode is ideal for recording risky headers that are not recorded unless explicitly stated. In the **Record Headers not in List** mode, VuGen inserts a `web_add_auto_header` function into your script for each of the unchecked headers that it detects during recording.

To determine which custom headers to record, you can perform a recording session indicating to VuGen to record all headers (see procedure below). Afterwards, you can decide which headers to record and which to exclude.

In this example, the *Content-type* header was specified in the **Record Headers in List** mode. VuGen detected the header and added the following statement to the script:

```
web_add_auto_header("Content-Type","application/x-www-form-urlencoded");
```

indicating to the server that the Content-type of the application is x-www-form-urlencoded.

To control the recording of custom headers:

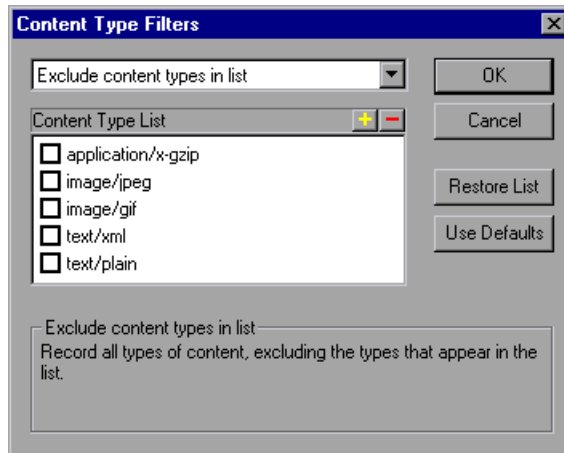
- 1** In the Recording Options tree, select the **Internet Protocol:Advanced** node.
- 2** Click **Headers**. The Headers dialog box opens.
- 3** Use one of the following methods:
 - To instruct VuGen not to record any Headers, choose **Do not Record Headers**.
 - To record only specific headers, select **Record Headers in list** and select the desired custom headers in the header list. Note that standard headers (such as *Accept*), are selected by default.
 - To record all headers, select **Record Headers not in list** and do not select any items in the list.
 - To exclude only specific headers, select **Record Headers not in list** and select the headers you want to exclude.
- 4** Click **Restore List** to restore the list to the corresponding default list. The **Record Headers in list** and **Record Headers not in list** each have a corresponding default list.

- 5 Click **OK** to accept the settings and close the Headers dialog box.

Filtering Content Type

VuGen allows you to filter the content type for your recorded script. You specify the type of the content you wish to record or exclude from your script. You can work in three modes: **Do not Filter Content Types**, **Exclude content types in list**, or **Exclude content types not in list**. When you work in the first mode, VuGen does not filter any content type. In the second mode, VuGen only excludes the selected content types. If you specify **Exclude content types not in list**, VuGen filters all content type except for the ones that are checked. By default, no filters are active.

For example, if you are only interested in the text and images on your Web site, you select **Exclude content types not in list** and specify the types *text/html*, *image/gif*, and *image/jpeg*. VuGen will record all HTML pages and images, and exclude resources such as *text/css*, *application/x-javascript* or other resources that appear on the site.



To filter content during recording:

- 1 In the Recording Options tree, select the **Internet Protocol:Advanced** node.
- 2 Click **Content Types**. The Content Type Filters dialog box opens.
- 3 Use one of the following methods:

- To instruct VuGen not to filter any content, choose **Do not Filter Content Types**.
 - To exclude only specific content types, select **Exclude content types in list** and select the desired content types from the list.
 - To include only specific content types, select **Exclude content types not in list** and select the content types you want to include.
- 4 Click **Restore List** to restore the list to the corresponding default list. The **Exclude content types in list** and **Exclude content types not in list** each have a corresponding default list.
 - 5 Click **OK** to accept the settings and close the Content Type Filters dialog box.

Specifying Non-Resource Content Types

When you record a script, VuGen indicates whether or not it will retrieve the resource during replay using the **Resource** attribute in the **web_url** function. If the **Resource** attribute is set to 0, the resource is retrieved during script execution. If the **Resource** attribute is set to 1, the Vuser skips the resource type.

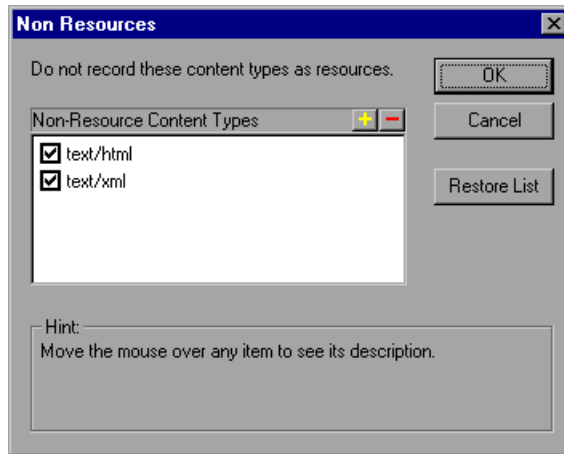
```
web_url("nav_tpo.gif",
        "URL=http://graphics.aa.com/images/navimg/nav_tpo.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=http://www.im.aa.com/American?BV_EngineID=...",
        "Mode=HTML",
        LAST);
```

You can exclude specific content types from being handled as resources. For example, you can indicate to VuGen that *gif* type resources should not be handled as a resource and therefore be downloaded unconditionally. When VuGen encounters a *gif* type resource, it sets the **Resource** attribute to 0, indicating to VuGen to download gifs unconditionally during replay.

To specify which content should not be recorded as resources:

- 1 In the Recording Options tree, select the **Internet Protocol:Advanced** node.

- 2 Click **Non-Resources** to open the dialog box and display the list of content types which should not be recorded as resources.



- 3 Click the “+” sign to add a content type to the list. Click the “-” sign to remove an existing entry.
- 4 Select the check boxes adjacent to the items you want to enable.
- 5 Click **Restore List** to restore the list to the default list.
- 6 Click **OK** to accept the settings and close the Non-Resources list.

35

Setting Recording Options for Web Vusers

Before recording a Web session, you can customize the recording options.

This chapter describes:

- ▶ Specifying which Browser to Use for Recording
- ▶ Selecting a Recording Mode
- ▶ Recording in HTML-Based Mode
- ▶ Recording in URL-Based Mode
- ▶ Specifying the Information to Record

The following information only applies to Web Vuser scripts.

About Setting Recording Options

VuGen enables you to generate Web Vuser scripts by recording typical processes that users perform on your Web site.

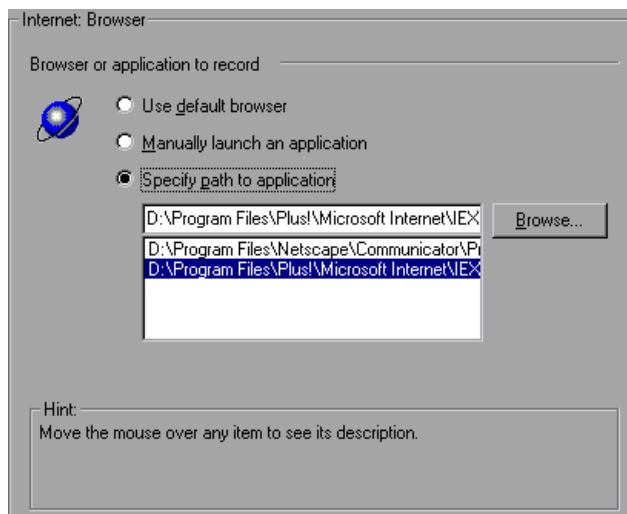
Before recording, you can configure the Recording Options and specify the information to record, the browser or client with which to record, and designate the content for your scripts.

You can set the common Internet protocol recording options, such as proxy settings and other advanced settings. For more information see Chapter 34, “Setting Recording Options for Internet Protocols.”

You can also set Correlation recording options for Web Vuser scripts. For more information, see Chapter 40, “Configuring Correlation Rules for Web Vuser Scripts.”

Specifying which Browser to Use for Recording

You can specify which browser VuGen uses when you record a Web Vuser script. You use the **Internet Protocol:Browser** node in the Recording Options tree to specify the location of the browser.



The following **Browser** options are available:

- **Use default browser**, to instruct VuGen to use the default Web browser on the recording computer.
- **Manually launch an application**, to instruct VuGen not to launch a browser when you start recording. You must manually launch a browser or application after you start the recording session.
- **Specify path to application**, to instruct VuGen to use the browser or application that you specify. Select a path from the list of paths, or click the **Browse** button to locate the required application.

Selecting a Recording Mode

VuGen lets you specify what information to record and which functions to use when generating a script by selecting a recording mode. The recording mode you select depends on your needs and environment. The available modes are *HTML-based script* and *URL-based script*.

The **HTML-based script** mode generates a separate step for each HTML user action. The recording of non-HTML elements depends on the settings for Handling of Non HTML-Generated Elements (see below).

The **URL-based script** recording mode captures all HTTP requests sent to the server as a result of user actions. This recording mode captures even non-HTML applications such as applets and non-browser applications. URL-based scripts are more scalable, contain more information about the user's actions, but are not as intuitive as the HTML-based scripts.

Follow these tips to decide which recording mode to choose.

- For browser applications, use the HTML-based mode.
- For non-browser applications, use the URL-based mode.

You can switch recording modes and advanced recording options while recording. The option of mixing recording modes is available for advanced users for performance tuning.

Recording in HTML-Based Mode

The **HTML-based** option, which is the default recording mode, instructs VuGen to record HTML actions in the context of the current Web page. It does not record all resources during the recording session, but downloads them during replay.

VuGen lets you set advanced options for HTML-based mode in the following areas:

- Script Types
- Handling of Non HTML-Generated Elements

Script Types

In HTML-based mode, you can specify the type of script:

- A script describing user actions
- A script containing explicit URLs only

The first option, **a script describing user actions**, is the default option. It generates functions that correspond directly to the action taken. It creates URL (**web_url**), link (**web_link**), image (**web_image**), and form submission (**web_submit_form**) functions. The resulting script is very intuitive and resembles a context sensitive recording.

```

/* HTML-based mode - a script describing user actions*/
...
web_url("Click Here For Additional Restrictions",
        "URL=http://www.im.aa.com/American...restrictions.html",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.im.aa.com/American?...",
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
...

```

The second option, **a script containing explicit URLs only**, records all links, images and URLs as **web_url** statements, or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions. The resulting script is less intuitive. This mode is useful for instances where many links within your site have the

same link text. If you record the site using the first option, it records an ordinal (instance) for the link, but if you record using the second option, each link is listed by its URL. This facilitates parameterization and correlation for that step.

The following segment illustrates a session recorded with **a script containing explicit URLs only** selected:

```

/* A HTML-based script containing explicit URLs only*/
...
web_url("Click Here For Additional Restrictions",
        "URL=http://www.im.aa.com/American...restrictions.html",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.im.aa.com/American?...
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

web_url("buttonhelp.gif",
        "URL=http://www.im.aa.com/American?BV_EngineID...",
        "TargetFrame=aamain",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.im.aa.com/American?...
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
...

```

Handling of Non HTML-Generated Elements

Many Web pages contain non-HTML elements, such as applets, XML, ActiveX elements, or javascript. These non-HTML elements usually contain or retrieve their own resources. For example, a javascript *js* file, called from the recorded web page, may load several images. An applet may load an external text file. Using the following options, you can control how VuGen records non HTML-generated elements.

The following options are available:

- Record within the current script step (default)
- Record in separate steps using concurrent groups
- Do not record

The first option, **Record within the current script step**, does not generate a new function for each of the non HTML-generated resources. It lists all resources as arguments of the `web_url` statement that was generated for the non-HTML element. The resources, arguments of `web_url`, are indicated by the `EXTRARES` flag. In the following example, the `web_url` function lists all of the non HTML-generated resources loaded on the page:

```
web_url("index.asp",
    "URL=http://www.daisy.com/index.asp",
    "TargetFrame=",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t2.inf",
    "Mode=HTML",
    EXTRARES,
    "Url=http://www.daisy.com/ScrollApplet.class", "Referer=", ENDITEM,
    "Url=http://www.daisy.com/board.txt", "Referer=", ENDITEM,
    "Url=http://www.daisy.com/nav_login1.gif", ENDITEM,
    ...
    LAST);
```

The second option, **Record in separate steps using concurrent groups**, creates a new function for each one of the non HTML-generated resources—it does not include them as items in the page's `web_url` function. All of the `web_url` functions generated for a resource, are placed in a concurrent group (surrounded by `web_concurrent_start` and `web_concurrent_end`).

In the following example, the above session was recorded with this option selected. A **web_url** function was generated for the applet and text file loaded with the applet:

```
web_url("index.asp",
        "URL=http://www.daisy.com/index.asp",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t2.inf",
        "Mode=HTML",
        LAST);

web_concurrent_start(NULL);
web_url("ScrollApplet.class",
        "URL=http://www.daisy.com/ScrollApplet.class",
        "Resource=1",
        "RecContentType=application/octet-stream",
        "Referer=",
        LAST);

web_url("board.txt",
        "URL=http://www.daisy.com/board.txt",
        "Resource=1",
        "RecContentType=text/plain",
        "Referer=",
        LAST);
web_concurrent_end(NULL);
```

The third option, **Do not record**, instructs VuGen not to record any of the resources generated by non-HTML elements.

Note that when you work in HTML-Based mode, VuGen inserts the **TargetFrame** attribute in the **web_url** statement. VuGen uses this

information to display the Web page correctly in the run-time browser and Test Result report.

```
web_url("buttonhelp.gif",
        "URL=http://www.im.aa.com/American?BV_EngineID=...",
        "TargetFrame=aamain",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.im.aa.com/American?BV_EngineID=...",
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
```

When you record the URL-based mode, VuGen records the content of all frames on the page and therefore omits the TargetFrame attribute.

Recording in URL-Based Mode

The **URL-based** mode option instructs VuGen to record *all* requests and resources from the server. It automatically records every HTTP resource as URL steps (**web_url** statements), or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions, nor does it record frames.

VuGen lets you set advanced option for this recording mode in the following area:

- Resource Handling
- Browser Cache

Resource Handling

In URL-based recording, VuGen captures all resources downloaded as a result of an HTTP request. By default, this option is enabled and VuGen records the resources in a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**), after the URL. Resources include files such as images, imported files, and *js* files. If you disable this option, the resources are listed as separate **web_url** steps, but not marked as a concurrent group.

The following segment illustrates a session recorded with the **Create concurrent groups for resources after their source HTML page** option disabled.

```

web_concurrent_start (NULL);
...
web_url("spacer.gif",
        "URL=http://graphics.aa.com/images/spacer.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=http://www.im.aa.com/American?BV_EngineID...",
        "Mode=HTTP",
        LAST);

web_url("calendar_functions.js",
        "URL=http://www.im.aa.com/travelp/calendar_functions.js",
        "Resource=1",
        "RecContentType=application/x-javascript",
        "Referer=http://www.im.aa.com/American?BV_Operation=...",
        "Mode=HTTP",
        LAST);
...
web_concurrent_end (NULL);

```

Note that the script includes *gif*, and *js* files. This mode also includes other graphic files and imported file such as *imp*, *txt*, or cascading style sheet (*css*) files.

Browser Cache

A browser cache stores recently viewed pages in the machine's memory, in order to reduce the time required to access the Web page. By default, the **Enable cache** option is disabled—VuGen retrieves all pages directly from the server and does not use the browser cache during recording.

Certain applications, however, will not be able to run without cache. To use the cache and only retrieve the newly-modified pages directly from the server, select the **Enable cache** option.

The **If-Modified-Since** HTTP header is a request by which the client checks whether a cached resource was modified on the server-side since the last

download. If the resource was modified, the client downloads it again to the cache. Otherwise, the server returns an HTTP status code of 304 —Not Modified. When cache is disabled, the **If-Modified-Since** header is suppressed and VuGen retrieves all pages directly from the server. In this mode, VuGen removes the **If-None-Match** request header in addition to the **Last-Modified**, **Expires** and **Etag** response headers. If the browser does not receive any of the above response headers, it does not store the image in the cache.

Note that you can manually control this header using the Advanced Header options (see “Recording Custom Headers” on page 437.)

Clearing the Browser Cache

By default, when the browser cache is enabled, VuGen clears the cache before recording. This means that it makes all of the items in the cache expired, so the browser must retrieve them directly from the server.

Clearing the cache requires VuGen to access all pages directly from their Web sites, even if the page had been recently accessed. If you are recording a Vuser that accesses a site repeatedly, you may choose not to clear the browser cache before recording.

To instruct VuGen not to clear the browser cache before recording, clear the **Clear cache before recording** check box. Note that this option only applies when recording with Internet Explorer.

Generating Custom Requests

When recording non-browser applications, you can instruct VuGen to record all HTTP requests as custom requests. VuGen generates a `web_custom_request` function for all requests, regardless of their content:

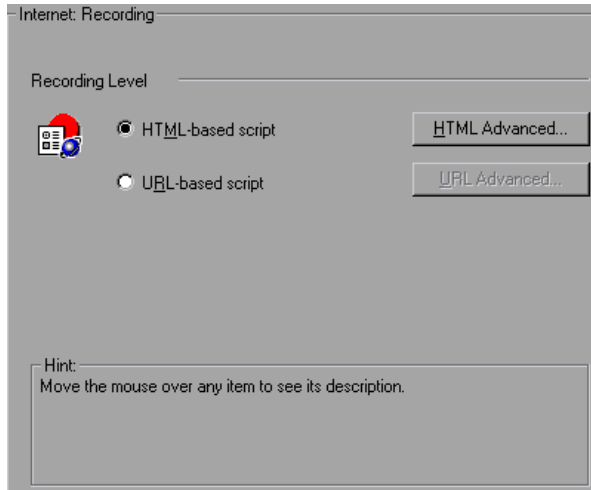
```
web_custom_request("www.aa.com",  
    "URL=http://www.aa.com/",  
    "Method=GET",  
    "Resource=0",  
    "RecContentType=text/html",  
    "Referer=",  
    "Snapshot=t1.inf",  
    "Mode=HTTP",  
    LAST);
```

Specifying the Information to Record

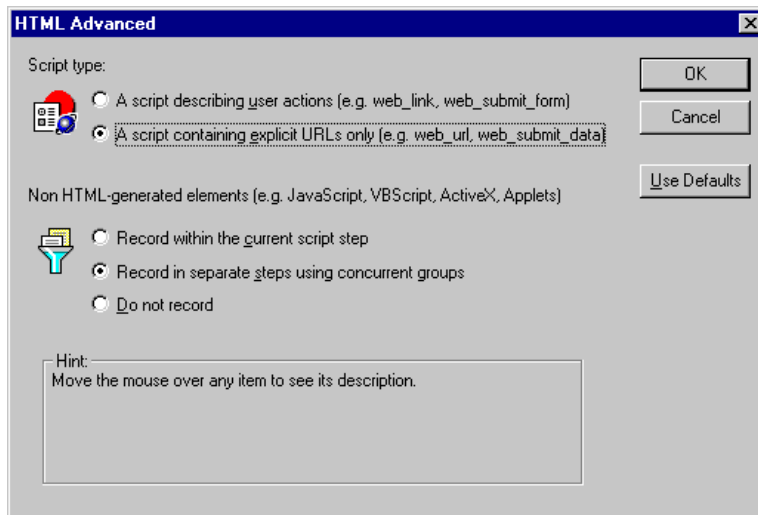
The previous section described the recording levels and their advanced options. This section describes the procedure for setting the options. Note that you can switch recording levels and advanced recording options while recording.

To set the recording options:

- 1 Choose **Tools > Recording Options** to open the Recording Options. Select the **Internet Protocol:Recording** node in the Recording Options tree.



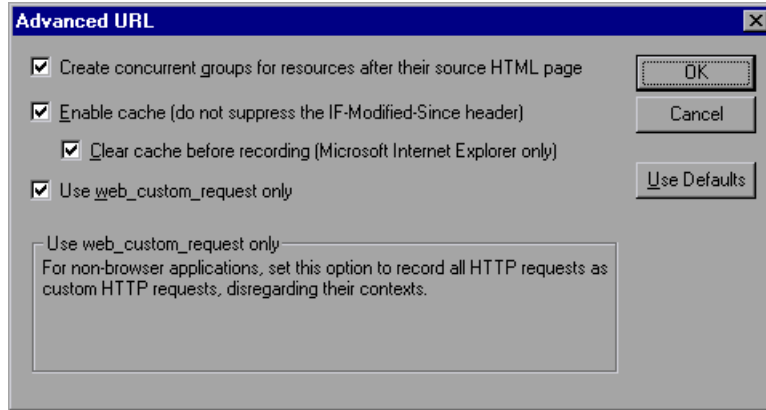
- 2 Select a recording mode: **HTML-based** or **URL-based**.
- 3 For HTML-based recording, click **HTML Advanced** to set additional options for script types and the handling of non-HTML elements.



Select a script type.

Select a method for handling non-HTML resources. (see “Recording in HTML-Based Mode,” on page 445).

- 4 For URL-based recording, click **URL Advanced** to set additional script options for resource handling and cache enabling.



Select **Create concurrent groups for resources after their source HTML page** to enable the recording of resources and marking them as a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**).

Select **Enable cache** to use the browser cache during recording. If you enable this option, clear the **Clear cache before recording** check box to instruct VuGen not to clear the cache and use previously accessed pages.

Select **Use web_custom_request only** to generate all HTTP requests as **web_custom_request** functions.

For more information about these options, see “Recording in URL-Based Mode,” on page 450.

36

Configuring Internet Run-Time Settings

After you record an Internet protocol Vuser script, you configure its run-time settings.

This chapter describes:

- ▶ Setting Proxy Options
- ▶ Setting Browser Emulation Properties
- ▶ Setting The Network Speed
- ▶ Setting Preferences

The following information applies to all Internet Protocol Vuser types such as Web, Wireless, Oracle NCA, and Real.

For information about the general run-time settings that apply to all Vusers, see Chapter 9, “Configuring Run-Time Settings.”

About Internet Run-Time Settings

After developing a Internet protocol Vuser script, you set the run-time settings. These settings let you configure your Internet environment so that Vusers can accurately emulate real users. You can set Internet run-time settings for Proxy, Browser, Speed Simulation, and other advanced preferences.

You set the Internet-related run-time settings from the Run-Time Settings dialog box. You click the appropriate node to specify the desired settings.

To display the Run-Time Settings dialog box:



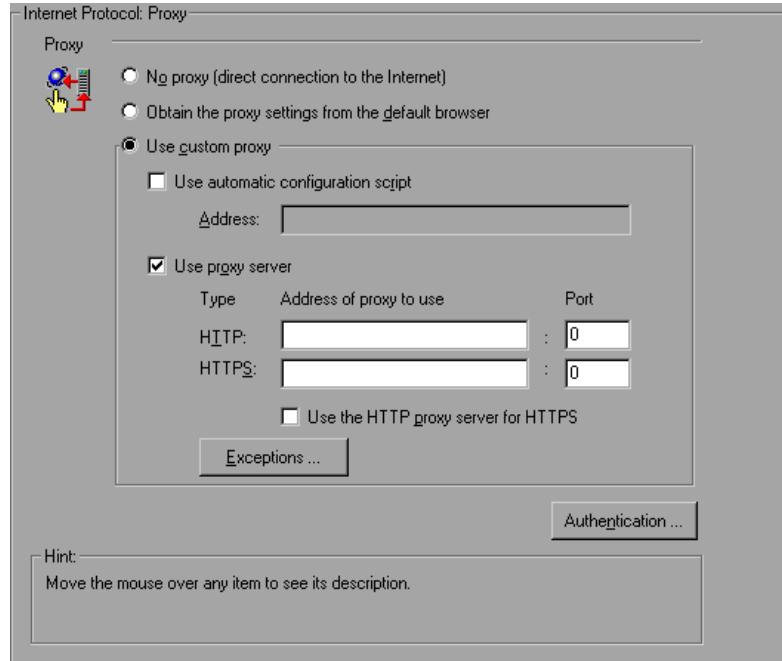
- Click the **Run-Time Settings** button on the VuGen toolbar.
- Press the keyboard shortcut key **F4**.
- Choose **Vuser > Run-Time Settings**.

Note that you can also modify the run-time settings from the ProTune Console.

Note: A run-time setting that is set by using Vuser functions overrides the corresponding setting set via the Run-Time Settings dialog box. For more information on using Vuser functions, see Chapter 32, “Using Web Vuser Functions.”

Setting Proxy Options

You use the **Internet Protocol:Proxy** node of the Run-Time Settings tree, to set the proxy-related settings:



By default, the Vuser uses the proxy settings of the browser used for recording in the Web recording options. It is recommended that you use the same settings for record and replay. For information about the Proxy Recording options, Chapter 35, “Setting Recording Options for Web Vusers.”

The following proxy options are available in the Run-Time settings.

- All Vusers always use direct connections to the Internet. This means that the connection is made without using a proxy server.
- All Vusers use the proxy settings of the default browser on the machine upon which they are running.
- All Vusers use a custom proxy server. You can supply the actual proxy server details, or the path of a proxy automatic configuration script (pac file in .js format) that enables automatic configuration. (see below)

To supply the details of the server, you specify its IP address or name and port. You can specify one proxy server for all HTTP sites, and another proxy server for all HTTPS (secure) sites.

- For URLs that you want Vusers to access directly, that is, without using the proxy server, click **Exceptions** and then supply the list of these URLs. In the Exceptions dialog box, you can also specify that the Vusers should not use a proxy server for local (intranet) addresses.
- If the proxy server requires authentication, click **Authentication**, and then supply the relevant password and user name.

Select the **Use the HTTP proxy server for HTTPS** check box to instruct the Vusers to use the HTTP proxy server for HTTPS sites, rather than specifying a specific server for secure sites.

Setting the Automatic Proxy Configuration

Automatic Proxy Configuration is a feature supported by most browsers. This feature allows you to specify a javascript file (with a .js extension) containing proxy assignment information. This script tells the browser when to access a proxy server and when to connect directly to the site, depending on the URL. In addition, it can instruct the browser to use a specific proxy server for certain addresses and another server for other addresses. To specify a configuration script in Internet Explorer (IE), choose **Tools > Internet Options**, and select the Connections tab. Click the **LAN Settings** button. In the LAN Settings dialog box that opens, you select the **Use automatic configuration script** option, and specify the location of the script. VuGen now supports automatic proxy configuration. You can specify a javascript for automatic proxy configuration, so that when ProTune runs the test, it uses the rules from the javascript file.

To track the behavior of the Vusers, generate a log during text execution and view the **Execution Log** tab or the *mdrv.log* file. The log shows the proxy servers that were used for each URL. In the following example, VuGen used

a direct connection for the URL `australia.com`, but the proxy server *aqua*, for the URL `http://www.google.com`.

```

Action1.c(6): t=1141ms: FindProxyForURL returned DIRECT
Action1.c(6): t=1141ms: Resolving australia.com
Action1.c(6): t=1141ms: Connecting to host 199.203.78.255:80
...
Action1.c(6): t=1281ms: Request done "http://australia.com/GetElement-
ByName.htm"

...
Action1.c(6): web_url was successful, 357 body bytes, 226 header bytes
Action1.c(15): web_add_cookie was successful
Action1.c(17): t=1391ms: FindProxyForURL returned PROXY aqua:2080
Action1.c(17): t=1391ms: Auto-proxy configuration selected proxy
aqua:2080
Action1.c(17): t=1391ms: Resolving aqua
Action1.c(17): t=1391ms: Connecting to host 199.203.139.139:2080
...l
Action1.c(17): t=1578ms: 168-byte request headers for "http://www.goo-
gle.com/" (RelFrameId=1)
Action1.c(17): GET http://www.google.com/ HTTP/1.1\r\n

```

To specify an automatic proxy configuration script in VuGen:

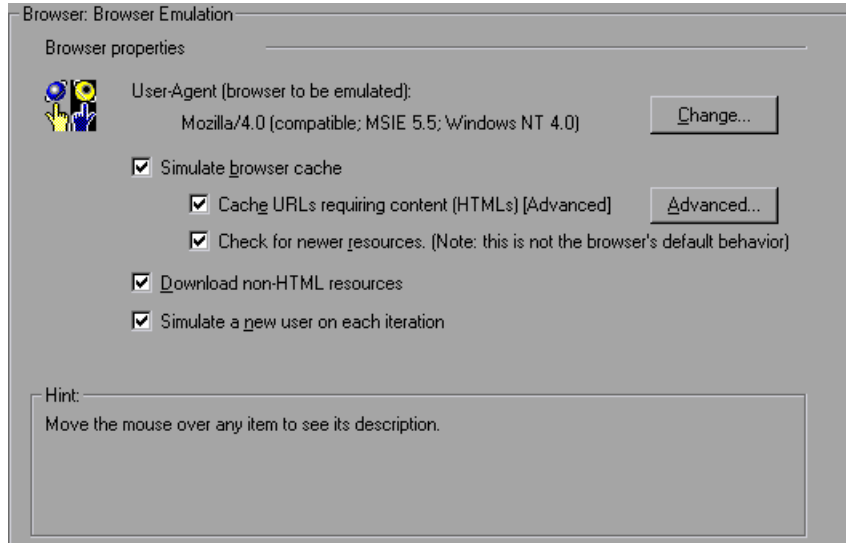
- 1 Open the Run-time settings (F4), and select the **Internet Protocol:Proxy** node.



- 2 Select the Use automatic configuration script option.
- 3 Specify the location of the javascript. You can specify either a web location beginning with `http://` (for example, `http://hostname/proxy.js`), or a location on the file server, for example, `C:\temp\proxy.js`.
- 4 Click **OK** to accept the settings.

Setting Browser Emulation Properties

You use the **Browser:Browser Emulation** node in the Run-Time Settings tree to set the browser and modem properties of your tuning environment.



Browser Properties

User-Agent browser to be emulated

Whenever a Vuser sends a request to a Web server, the request always includes an HTTP header. The first line of text contains a verb (usually "GET" or "POST"), the resource name (for example "pctl/default.htm"), and the version of the protocol (for example "HTTP/1.0"). Subsequent lines contain "header information" in the form of an attribute name, a colon, and some value. The request ends with a blank line.

All Internet Vuser headers include a **User-Agent** header that identifies the type of browser (or toolkit for Wireless) that is being emulated. For example,

User-Agent: Mozilla/3.01Gold (WinNT; I)

identifies the Browser as Netscape Navigator Gold version 3.01 running under Windows NT on an Intel machine.

Click **Change...** to specify the browser information to include in the header. You can specify that a Web Vuser emulate any of the standard browsers. Alternatively, for non-browser HTTP applications, you can specify the HTTP client to match a specific user's application. In this case, you must supply a **Custom User Agent** string that is included in all subsequent HTTP headers. By default, the Vuser uses the Internet Explorer 4.0 browser agent.

Note that this option does not indicate the replay browser—the browser through which the script is played. It only effects the **User-Agent** attribute of the HTTP Header sent to the server.

Simulate browser cache

This option instructs the Vuser to simulate a browser with a cache. A cache is used to keep local copies of frequently accessed documents and thereby reduces the time connected to the network. By default, cache simulation is enabled. When the cache is disabled, ProTune still downloads each page image only once. When running multiple Vusers from the Console, every Vuser uses its own cache and retrieves images from the cache. If you disable this option, all Vusers emulate a browser with no cache available.

You can modify your Run-Time settings to match your browser settings for Internet Explorer:

Browser Setting	Run-Time Setting
Every visit to the page	Select Simulate Browser Cache and enable Check for newer pages when simulating browser cache .
Every time you start Internet Explorer	Select Simulate Browser Cache only
Automatically	Select Simulate Browser Cache only
Never	Select Simulate Browser Cache and disable Check for newer pages when simulating browser cache .

Note: Unlike a regular browser cache, the cache assigned to a Vuser simulates storage of graphic files only. The cache does not store text or other page contents associated with the Web page.

You can also set the following two browser cache options:

Cache URLs requiring content (HTML): This option instructs VuGen to cache only the URLs that require the HTML content. The content may be necessary for parsing, verification, or correlation. When you select this option, HTML content is automatically cached. To define additional content types whose content you want to cache, click **Advanced**. (This increases the memory footprint of the virtual user.) This option is disabled by default.

In the Advanced dialog box, enable the **Specify URLs requiring content in addition to HTML page** option, and click the plus sign to add additional content types, such as text/plain, text/xml, image/jpeg, and image/gif.

Check for newer resources when simulating browser cache: This settings instructs the browser to check for newer resources than those stored in the cache, for the specified URL. This is done by adding the “If-modified-since” attribute to the HTTP header. This option guarantees that the most recent version of the page always appears, but it generates more traffic during the session. By default, browsers do not check for newer resources, and therefore this option is disabled. Configure this option to match the settings in the browser that you wish to emulate.

Download non-HTML resources

This option instructs Vusers to load graphic images when accessing a Web page during replay. This includes both graphic images that were recorded with the page, and those which were not explicitly recorded along with the page. When real users access a Web page, they wait for the images to load. Therefore, enable this option if you are trying to test the entire system, including end-user time (enabled by default). To increase performance and not emulate real users, disable this option.

Note: Disable this option if you experience discrepancies in image checks, since some images vary each time you access a Web page (for example, advertiser banners).

Simulate a new user each iteration

Instructs VuGen to reset all HTTP contexts between iterations to their states at the end of the *init* section. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. It deletes all cookies, closes all TCP connections (including keep-alive), clears the emulated browser's cache, resets the HTML frame hierarchy (frame numbering will begin from 1) and clears the user-names and passwords. This option is enabled by default.

Performing HTML Compression

Browsers that support HTTP 1.1 can decompress HTML files. The server compresses the files for transport, substantially reducing the bandwidth required for the data transfer.

To enable compression in VuGen, add the function

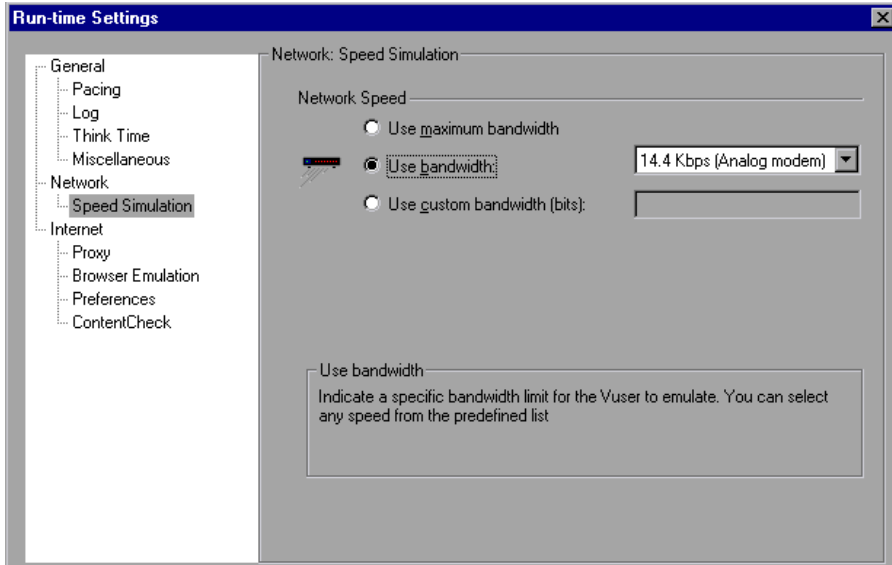
```
web_add_auto_header("Accept-Encoding", "gzip");
```

at the beginning of the script. To verify that the server sent compressed data, search for the string *Content-Encoding: gzip* in the section of the server's responses of the Execution log. The log also shows the data size before and after decompression.

Compression has a greater effect on large data transfers—the larger the data, the greater effect the compression will have. When working with larger data, you can also increase the network buffer size (see “Network Buffer Size” on page 471) to get the data in single chunks.

Setting The Network Speed

You use the **Network:Speed Simulation** node in the Run-Time Settings tree, to set the modem emulation for your tuning environment.



Network Speed

Select a bandwidth that best emulates the environment under test.

Use maximum bandwidth

By default, bandwidth emulation is disabled and the Vusers run at the maximum bandwidth that is available over the network.

Use bandwidth ...

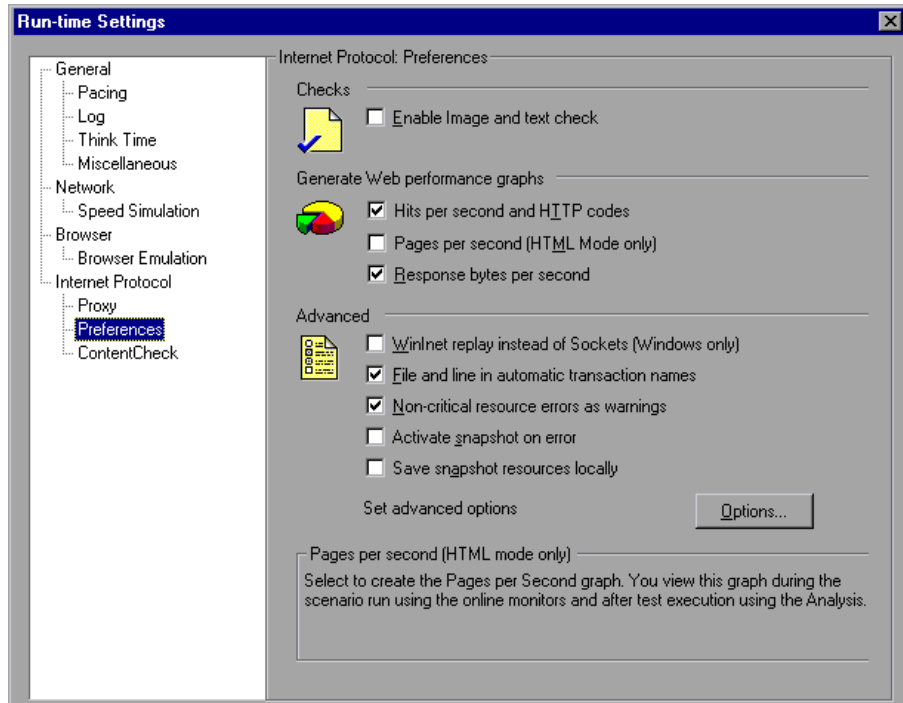
Indicate a specific bandwidth level for your Vuser to emulate. You can select a speed ranging from 14.4 to 512 Kbps, emulating analog modems, ISDN, or DSL.

Use custom bandwidth ...

Indicate a bandwidth limit for your Vuser to emulate. Specify the bandwidth in bits, where 1 Kilobit=1024 bits.

Setting Preferences

You use the **Internet Protocol:Preferences** node in the Run-Time Settings tree, to set the settings related to checks, graph information, and advanced settings.



Checks

The **Enable image and text checks** check box allows the Vuser to perform verification checks during replay, by executing the verification functions: **web_find** or **web_image_check**. This option only applies to statements recorded in HTML-based mode. Users running with verification checks use more memory than Vusers who do not perform checks (disabled by default).

Generate Web Performance Graphs

Instructs a Vuser to collect data used to create Web Performance graphs. You view the Hits per Second, Pages per Second and Response Bytes per Second graphs during test execution using the online monitors and after test

execution using the Analysis. You view the Component Breakdown graph after test execution using the Analysis. Select the types of graph data for the Vuser to collect.

Note: If you do not use the Web performance graphs, disable these options to save memory.

Advanced

WinInet Replay (instead of Sockets)

Instructs VuGen to use the WinInet replay engine. VuGen has two HTTP replay engines: Sockets-based (default) or WinInet based. The WinInet is the engine used by Internet Explorer and it supports all of the features incorporated into the IE browser. The limitations of the WinInet replay engine are that it is not scalable, nor does it support UNIX. In addition, when working with threads, the WinInet engine does not accurately emulate the modem speed and number of connections.

VuGen's proprietary sockets-based replay is a lighter engine that is scalable for testing. It is also accurate when working with threads. The limitation of the sockets-based engine is that it does not support SOCKS proxy. If you are recording in that type of environment, use the WinInet replay engine.

File and line in automatic transaction names

Creates unique transaction names for automatic transactions by adding file name and line number to the transaction name (enabled by default).

Note: This option places additional information in the log file, and therefore requires more memory.

Non-critical item errors as warnings

This option returns a warning status for a function which failed on an item that is not critical, such as an automatically downloaded image that failed to

load, or a Java applet that failed to start. This setting is primarily for advanced users who have determined that a non-critical error should fail in their environment (warning enabled by default). You can set a content-type to be critical by adding it to the list of Non-Resources. For more information, see the “Specifying Non-Resource Content Types,” on page 440.

Activate Snapshot on Error

This option generates a snapshot when an error occurs. You can view the snapshot in the Console by showing the Vuser Log and double-clicking on the line at which the error occurred.

Save a local copy of all snapshot resources during replay

Instructs VuGen to save the snapshot resources to files on the local machine. This feature lets the Run-Time viewer create snapshots more accurately and display them quicker.

Additional Options

You can set the following advanced options for the Internet Preferences: DNS caching, HTTP version, Keep-Alive HTTP connections, HTTP-request connect timeout, HTTP-request receive timeout, Network buffer size, and Step download timeout.

DNS caching

Instructs the Vuser to save a host's IP addresses to a cache after resolving its value from the Domain Name Server. This saves time in subsequent calls to the same server. In situations where the IP address changes, as with certain load balancing techniques, be sure to disable this option to prevent Vuser from using the value in the cache. (enabled by default)

HTTP version

Specifies which version HTTP to use: version 1.0 or 1.1. This information is included in the HTTP request header whenever a Vuser sends a request to a Web server. The default is HTTP 1.1. HTTP 1.1 supports the following features:

- Persistent Connections—see “Keep-Alive HTTP connections” on page 470.

- HTML compression—see “Performing HTML Compression” on page 465.
- Virtual Hosting—multiple domain names sharing the same IP address.

Keep-Alive HTTP connections

Keep-alive is a term used for an HTTP extension that allows persistent or continuous connections. These long-lived HTTP sessions allow multiple requests to be sent over the same TCP connection. This improves the performance of the Web server and clients.

The keep-alive option works only with Web servers that support keep-alive connections. This setting specifies that all Vusers that run the Vuser script have keep-alive HTTP connections enabled. (enabled by default)

Step timeout caused by resources is a warning

Issues a warning instead of an error when a timeout occurs due to a resource that did not load within the timeout interval. For non-resources, VuGen always issues an error. (disabled by default)

Parse HTML content-type

When expecting HTML, parse the response only when it is the specified content-type: **HTML**, *text/html*, **TEXT** any text, or **ANY**, any content-type. Note that text/xml is not parsed as HTML. The default is **TEXT**.

HTTP-request Connect Timeout (seconds)

The time, in seconds, that a Vuser will wait for the connection of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. The default value is 120 seconds.

HTTP-request Receive Timeout (seconds)

The time, in seconds, that a Vuser will wait to receive the response of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. The default value is 120 seconds.

The timeout settings are primarily for advanced users who have determined that acceptable timeout values should be different for their environment. The default settings should be sufficient in most cases. If the server does not

respond in a reasonable amount of time, check for other connection-related issues, rather than setting a very long timeout which could cause the scripts wait unnecessarily.

Step download timeout (seconds)

The time that the Vuser will wait before aborting a step in the script. This option can be used to emulate a user behavior of not waiting for more than x seconds for a page.

Network Buffer Size

Sets the maximum size of the buffer used to receive the HTTP response. If the size of the data is larger than the specified size, the server will send the data in chunks, increasing the overhead of the system. When running multiple Vusers from the Console, every Vuser uses its own network buffer. This setting is primarily for advanced users who have determined that the network buffer size may affect their script's performance. The default is 12K bytes.

Fixed think time upon authentication retry (seconds)

Automatically adds a think time to the Vuser script for emulating a user entering authentication information (username and password). This think time will be included in the transaction time. The default is 0.

Obtaining Debug Information

When you run a Vuser script, the execution information is displayed in the Console. You control the amount of information sent using the **Log** node of the General run-time settings. (see “Configuring the Log Run-Time Settings” on page 125)

Debug information includes:

- log information
- transaction failures
- the connection status with the gateway—connecting, disconnecting, and redirecting. (WAP only)

To obtain more information for debugging, edit the *default.cfg* file. Locate the **WEB** section and set the **LogFileWrite** flag to *1*. The resulting trace file documents all events in the execution of the script.

37

Checking Web Page Content

After you record a Web Vuser script, you can configure run-time settings to check the page content.

This chapter describes:

- ▶ Setting the ContentCheck Run-Time Settings
- ▶ Defining a ContentCheck Rule

The following information only applies to Web Vuser types.

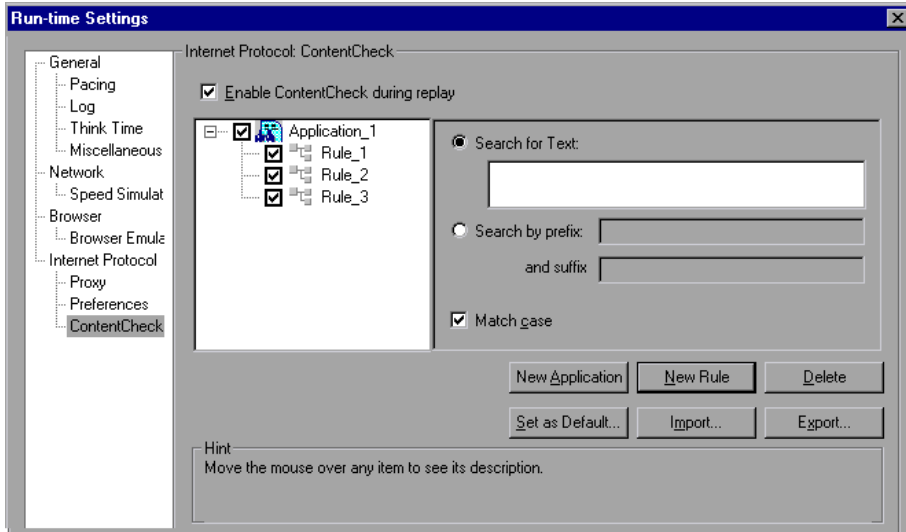
About Checking Web Page Content

VuGen's Content Check mechanism allows you to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text *ASP Error*. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay. Note that VuGen searches the body of the pages—not the headers.

Setting the ContentCheck Run-Time Settings

You use the **Internet Protocol:ContentCheck** Run-Time setting to define the string for which to search.



Click the **New Application** button to add a new application to the list of applications whose content to check. Note that even after you define applications, you can instruct VuGen to disable them for a specific test run, by clearing the **Enable ContentCheck during replay** option. You can also disable a specific rule by clearing the check box adjacent to it.

Use the **New Rule** button to add rules for existing applications. You specify text, a prefix and suffix, and case sensitivity. Each application server may have one or more rules.

VuGen lets you export and import Content Check to and from XML files. This capability lets you apply the rules from one script, to another one.

There are three types of rules for Content Checks: Installation, Default, and per script. *Installation* rules are provided automatically during installation of the product. *Default* rules, apply to all scripts executed on your machine. The *per script* rules are the ones defined for the current script. (When you modify or add rules in the run-time settings, these changes only apply to the

current script.) To instruct VuGen to add a rule to the list of Default rules so that it will apply to all scripts on that machine, click **Set as Default**.

When working on multiple scripts, or when performing a product upgrade, a conflict may arise between the default rules and the script rules. VuGen asks you if you want to merge the rules. When you merge the rules (recommended), the rule is added to the list of rules for the application.

The rules are stored in standard xml files. You can export your rule files and share them with other users or import them to other machines.

To use the default settings for all of your applications, click **Use Defaults** which imports the definitions from the Defaults file. It opens a dialog box with a list of the applications and their default settings. You can choose to import these definitions or modify them. If this conflicts with one of the rules, VuGen issues a warning indicating that it is a Conflicting Rule. You can merge or overwrite the rules defined in the Defaults file with the active ones.

Defining a ContentCheck Rule

You use the **Internet Protocol:ContentCheck** node in the Run-Time Setting tree, to define the rules for checking Web page content.

To define a ContentCheck rule:

- 1** Open the Run-Time settings and select the **Internet Protocol:ContentCheck** node.
- 2** Enable or disable the relevant applications and rules, by clearing or selecting the check boxes adjacent to the rule or application in the left pane.
- 3** To search for the actual text string, select **Search for Text** and specify the text for which you want to search. It is recommended that you be as specific as possible. For example, do not use the term *Error*, rather *ASP Error* or text specific to the application.
- 4** To search for the text preceding and following your string, select **Search by Prefix** and specify the prefix and suffix.
- 5** To indicate a case sensitive search, select the **Match case** check box.

- 6** To set a rule as a default, indicating that it should apply to all scripts on that machine, select the rule or application and click **Set as Default**.
- 7** To export the rule file click **Export** and specify a save location.
- 8** To import a rule file, click **Import** and locate the file.
- 9** To remove an application or rule, select it and click **Delete**.
- 10** To use the default settings for all of your applications, click **Use Defaults**. A dialog box opens with a list of the applications and their default settings. You can choose to overwrite or merge the rules if there are conflicts.

38

Verifying Web Pages Under Load

You can add Web 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 describes:

- ▶ Adding a Text Check
- ▶ Using Other Text Check Methods
- ▶ Adding an Image Check
- ▶ Defining Additional Properties
- ▶ Using Regular Expressions

The following information only applies to Web Vuser scripts.

About Verification Under Load

VuGen enables you to add Web 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, an image, or a Java applet.

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?

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. In the script segment below, the Vuser accesses the hypertext link on the site *www.aa.com*.



The browser displays the page associated with this URL. The Vuser then executes a text check on this Web page. For example, if the word *Specials* appears on the page, the check passes. If *Specials* does not appear because, for example, the correct page was not returned by the server, the check fails.

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. It is generally more convenient to define checks while recording—when the Web page that you want to check is visible.

When you add a Web check, VuGen adds a Web check icon to the current step in the tree view of the Vuser script. Web check icons are always indented under the associated step. When you run the Vuser script, VuGen conducts the check on the Web page that is displayed after the step is executed.

Note: A Vuser conducts Web checks during script execution only if checks are enabled, and if the script runs in HTML mode. To enable checks, select the **Enable image and text check** option in the **Preferences** tab in the Run-Time Settings dialog box. For details, see Chapter 9, “Configuring Run-Time Settings.”

VuGen uses several different Web check icons, each one representing a different check type:

Web Check Icon	Description
Text 	A text check is a search for a specified text string on a Web page.
Image 	An image check is a search for a specified image on a Web page.

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.

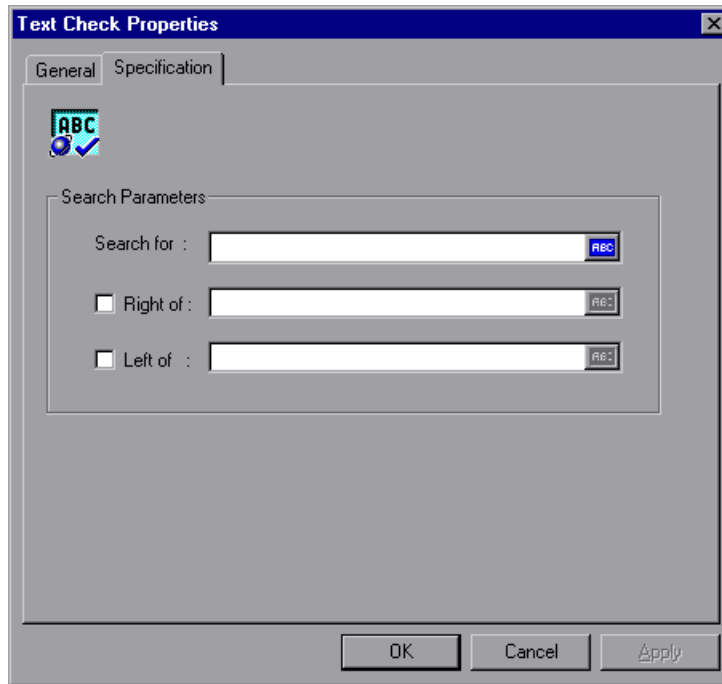
To add a text check after recording:

- 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.

Note: During a Web browser recording session, the VuGen main window may be minimized. To add a text check during recording, restore the VuGen main window.

- 2** Expand **Web Checks** in the **Step Type** tree.

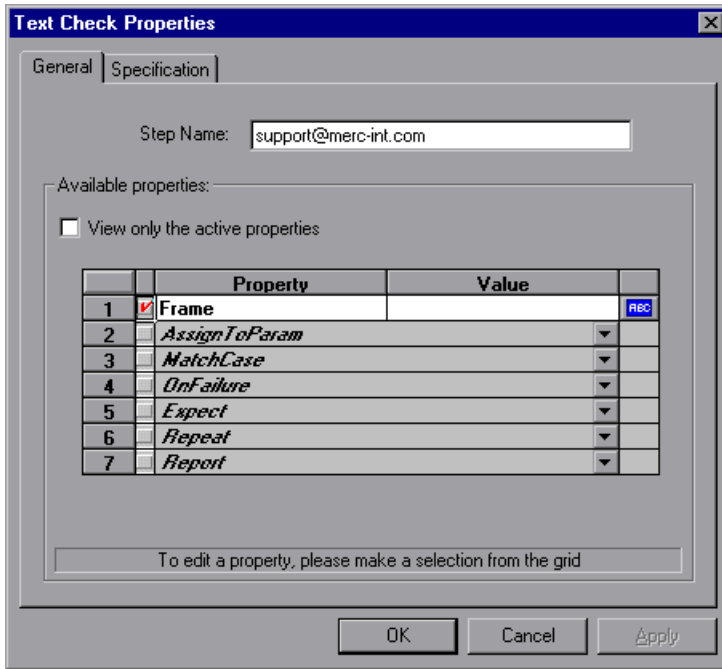
- 3 Select **Text Check**, and click **OK**. The Text Check Properties dialog box opens. Ensure that the **Specification** tab is visible.



- 4 In the **Search for** box, type the string whose presence you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see Chapter 7, "Defining Parameters."
- 5 To specify the position of the search string relative to adjacent text, select the **Right of** or **Left of** check box. Then type the text in the appropriate field. For example, to verify that the string "support@mercuryinteractive.com" appears to the right of the word "e-mail:," select **Right of** and then type "e-mail:" in the **Right of** box.

An ABC icon indicates that the string in the **Right of** or **Left of** box has not been assigned a parameter. For details on assigning parameters, see Chapter 7, "Defining Parameters."

- 6 Name the text check. Click the **General** tab and type a name for the text check in the **Step Name** box. Use a name that will make the check easy to identify later on.



- 7 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 488.



- 8 Click **OK** to accept the settings. An icon representing the new text check is added to the associated step in the script. In script view, the, Text Check icon appears as a **web_find** function.



To add a text check during recording:

- 1 Using the mouse, mark the desired text.
- 2 Click the **Insert text check** icon on the recording toolbar.



Using Other Text Check Methods

In addition to the `web_find` function, you can use two other enhanced functions to search for text within an HTML page:

- `web_reg_find`
- `web_global_verification`

The `web_reg_find` function is a registration type function. It 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.

This function differs from the `web_find` function, in that it is not limited to an HTML-based script (see **Recording Options > Recording** tab). This function also has additional 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 following attributes are available for `web_reg_find`:

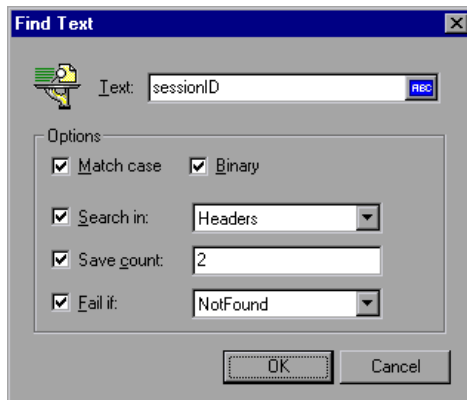
- **Text:** The text string to search for. This attribute must be a non-empty, null-terminated character string. The search mechanism is case sensitive; to ignore the case, add `/IC` after the boundary. Specify `/BIN` (or check the **Binary** check box in the step's properties) after the text to specify binary data. Use the format `"Text=string"`.

Instead of specifying **Text**, you can specify the following two attributes:

- **TextPfx:** The prefix of the text string for which you are searching. To ignore the case, add `/IC` after the boundary. Specify `/BIN` after the text to specify binary data. Use the format `"TextPfx=string"`.

- ▶ **TextSfx:** The suffix of the text string for which you are searching. To ignore the case, add "/IC" after the boundary. Specify "/BIN" after the text to specify binary data. Use the format "TextSfx=string".
- ▶ **Search:** Where to search for the text. The available values are Headers, BODY, NORESOUCE, or ALL. The default is BODY. Use the format "Search=value". (optional)
- ▶ **SaveCount:** The number of matches that were found. To use this attribute, Specify SaveCount=param_name where param_name is the variable to which a null-terminated ASCII value is stored. (optional)
- ▶ **Fail:** The handling method when the string is not found. The available values are **Found**, **NotFound**, and **None**. *Found* indicates that a failure occurs when the text is found (e.g. "Error"). *Not Found* indicates that a failure occurs when the text is not found. When the **SaveCount** attribute is specified, the default is None-no failure. When the **SaveCount** attribute is omitted, the default is NotFound. Note that you cannot explicitly assign the value None to the **Fail** attribute.

You can also set the above attributes from the functions Properties dialog box. Enter the tree view, select the function, and choose **Properties** from the right-click menu. A dialog box opens in which you can enter all of the argument values.



In the following example, **web_reg_find** function searches for the text string "Welcome". If the string is not found, it fails and the script execution stops.

```
web_reg_find("Text=Welcome", "Fail=Found", LAST);
web_url("Step", "URL=...", LAST);
```

The **web_global_verification** function allows you to search the data of an entire business process. In contrast to **web_reg_find**, which only applies to the next Action function, this function applies to *all* subsequent Action functions such **web_url**. By default, the scope of the search is NORESOURCE, searching only the HTML body, excluding headers and resources. The **web_global_verification** function is ideal for detecting application level errors that are not included the HTTP status codes. This function is not limited to an HTML-Based script (see **Recording Options > Recording tab**).

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.

You can add user-defined image checks either during or after recording. After recording, you can edit any existing image checks in your script.

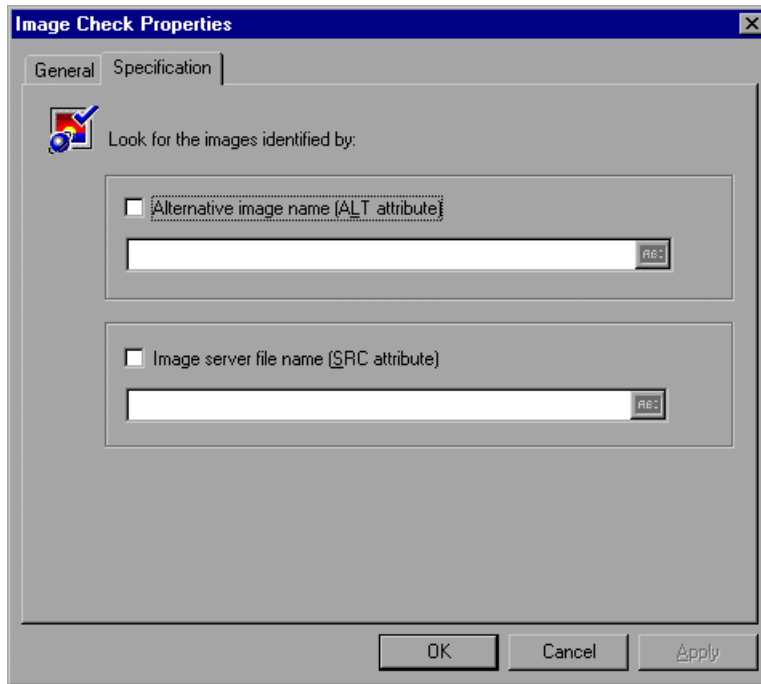
To add an image check:

- 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.

Note: During a Web browser recording session, the VuGen main window may be minimized. To add an image check during recording, restore the VuGen main window.

- 2 Expand **Web Checks** in the **Step Type** tree.

- 3 Select **Image Check**, and click **OK**. The Image Check Properties dialog box opens. Ensure that the **Specification** tab is visible.

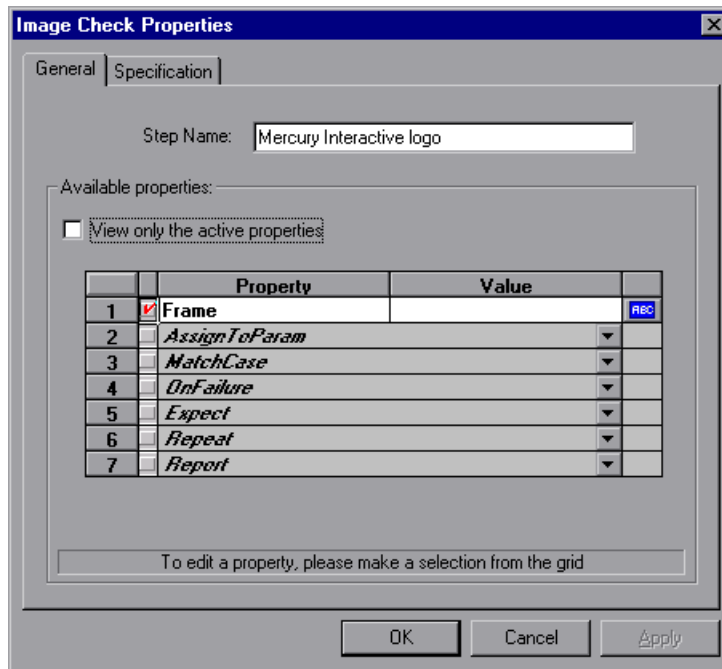


- 4 Select a method to identify the image:
 - To identify the image using its ALT attribute, select the **Alternative image name (ALT attribute)** check box, and type the ALT attribute. When you run the script, the Vuser searches for an image that has the specified ALT attribute.
 - To identify the image using the SRC attribute, select the **Image server file name (SRC attribute)** check box, and type the SRC attribute. When you run the script, the Vuser searches for an image that has the specified SRC attribute.

An ABC icon indicates that the ALT or SRC attribute has not been assigned a parameter. For details on assigning parameters, see Chapter 7, "Defining Parameters."

Note: If you select both the ALT attribute and SRC attribute check boxes, the Vuser searches for an image that has both the specified ALT attribute and the specified SRC attribute.

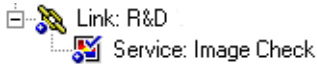
- 5 To name the image check, click the **General** tab. In the **Step Name** box, type a name for the image check. Use a name that will make the check easy to identify later on.



- 6 The properties table displays additional properties that define the check. Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column. For details on assigning property values, see “Defining Additional Properties” on page 488.



- 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.

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. Ensure 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.
 - **MatchCase:** Select **YES** to conduct a case-sensitive search. Select **NO** to conduct a non-case-sensitive search. The default value is **NO**.
 - **OnFailure:** Select **Abort** to abort the entire Vuser script if the check fails. VuGen aborts the Vuser script regardless of the error-handling method that has been set in the run-time settings. Select **Continue** to have the error-handling method defined in the run-time settings determine whether or not the script is aborted if the check fails. The default value is **Continue**. For details on defining the error handling method, see Chapter 9, “Configuring Run-Time Settings.”
 - **Expect:** Select **NotFound** to indicate that the check is successful if the Vuser does not find the specified check object. Select **Found** to indicate

that the check is successful if the Vuser finds the specified check object. The default value is **Found**.

- **Repeat:** Select **YES** to search for multiple occurrences of the specified check object. Select **NO** to end the check as soon as one occurrence of the specified check object is found. The Vuser script continues with the next step. This option is useful when searching through a Web page that may have multiple occurrences of the check object. The default value is **YES**.
- **Report:** Select **Always** to always view a detailed description of the check results in the Execution Log. Select **Failure** to view detailed check results only when the check fails. Select **Success** to view detailed check results only when the check succeeds. The default value is **Always**.
- **Inside:** The scope within the HTML page in which to perform a search: list (inside a list or combo box) or link (the text of a link).

An ABC icon indicates that the property value has not been assigned a parameter. Click the icon to assign a parameter. For more information, see Chapter 7, “Defining Parameters.”

Using Regular Expressions

When adding a text check, you can specify the value type as a *regular expression*. Using a regular expression increases the flexibility and adaptability of a check.

In a regular expression, any character that is not one of the special characters described below is searched for literally. When a special character is preceded by a backslash (\), the Vuser searches for the literal character.

The following options can be used to create regular expressions:

- Matching Any Single Character
- Matching Any Single Character within a Range
- Matching One or More Specific Characters

Matching Any Single Character

A question mark (?) instructs VuGen to search for any single character. For example,

welcome?

matches welcomes, welcomed, or welcome followed by a space or any other single character. A series of question marks indicates a sequence of unspecified characters. The length of the sequence is equal to the number of periods.

Matching Any Single Character within a Range

In order to match a single character within a range, you can use square brackets ([]). For example, to search for a date that is either 1968 or 1969, write:

196[89]

You can use a hyphen (-) to indicate an actual range. For instance, to match any year in the 1960s, write:

196[0-9]

A hyphen does not signify a range if it appears as the first or last character within brackets, or after a caret (^).

A caret (^) instructs VuGen to match any character except for the ones specified in the string. For example:

[^A-Za-z]

matches any non-alphabetic character. The caret has this special meaning only when it appears first within the brackets.

Note that within brackets, the following characters are literal:

period (.)
asterisk (*)

left bracket ([)

backslash (\)

If the right bracket is the first character in the range, it is also literal. For example:

`[]g-m]`

matches the right bracket, and g through m.

Matching One or More Specific Characters

An asterisk (*) instructs VuGen to match zero or more occurrences of the preceding character. For example:

`Q*`

causes VuGen to match Q, QQ, QQQ, and so forth.

39

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, rename, and delete existing steps.

This chapter describes:

- Adding a Step to a Vuser Script
- Deleting Steps from a Vuser Script
- Modifying Action Steps
- Modifying Control Steps
- Modifying Service Steps
- Modifying Web Checks (Web only)

The following information applies to Web and Wireless Vuser scripts.

About Modifying Web and Wireless Vuser Scripts

After recording a browser or toolkit 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 31, "Creating Web Vuser Scripts."

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.

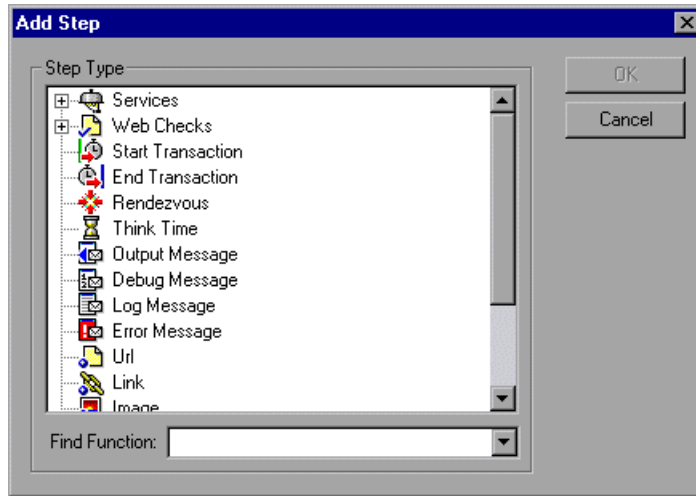
Adding a Step to a Vuser Script

In addition to the steps that VuGen records during the browser or toolkit 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 a(n)...	See...
ProTune function	Chapter 6, “Enhancing Vuser Scripts”
Service step	“Modifying Service Steps” on page 513
Web Check	“Modifying Web Checks (Web only)” on page 514
Transaction	“Modifying a Transaction” on page 510
Rendezvous point	“Modifying a Rendezvous Point” on page 512
Think time step	“Modifying Think Time” on page 512

To add a(n)...	See...
URL step	"Modifying a URL Step" on page 497
Link step	"Modifying a Hypertext Link Step (Web only)" on page 499
Image step	"Modifying an Image Step (Web only)" on page 500
Submit form step	"Modifying a Submit Form Step (Web only)" on page 502
Submit data step	"Modifying a Submit Data Step" on page 505
Custom request step	"Modifying a Custom Request Step" on page 508
User-defined step	Chapter 6, "Enhancing Vuser Scripts"

Deleting Steps from a Vuser Script

After recording a browser or toolkit 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 “Selecting a Recording Mode,” on page 445.

Setting the Replay Mode



In the URL step’s Properties dialog box, you can modify the mode settings to instruct ProTune to execute the script in a mode other than the recorded mode. To customize the replay mode, select the **Record mode** check box. The available replay modes are:

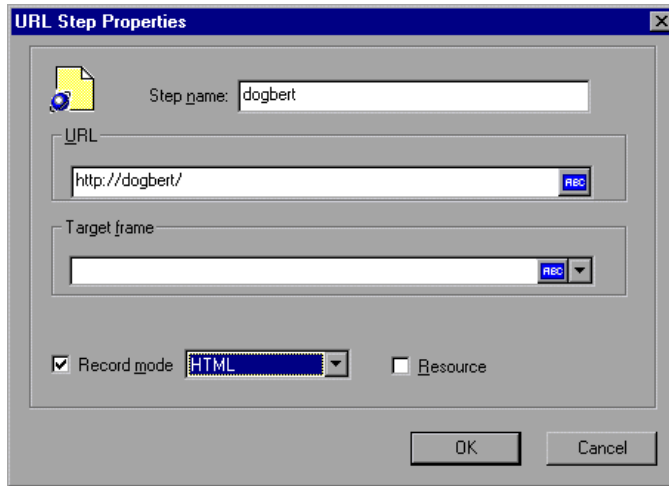
HTML: Automatically download all resources and images and store the required HTTP information for the steps that follow. This is ideal for script with Web links.

HTTP: Do not download any resources for this step during replay. Download only resources that are explicitly represented by functions.

You can also indicate that a certain step should not be counted as a resource. For example, if you have a step that represents a specific image that you want to skip, you can instruct VuGen to exclude that resource type. For more information, see the “Resource Handling,” on page 450.

To modify the properties of a URL step:

-  **1** In the tree view of the Vuser script, select the URL step you want to edit. URL steps are shown using the URL icon.
-  **2** Click the **Properties** button on the VuGen toolbar. The URL Step Properties dialog box opens.



- 3** To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.
- 4** In the **URL** box, type the Web address (URL) of the Web page that is accessed by the URL step. An *ABC* icon indicates that the URL has not been assigned a parameter. For details on assigning parameters, see Chapter 7, “Defining Parameters.”
- 5** In the **Target frame** list, select one of the following values:
 - _TOP:** replaces the whole page
 - _BLANK:** opens a new window
 - _PARENT:** replaces the parent of the last (changed) frame
- 6** To customize the replay mode, select the **Record mode** check box. Choose 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)

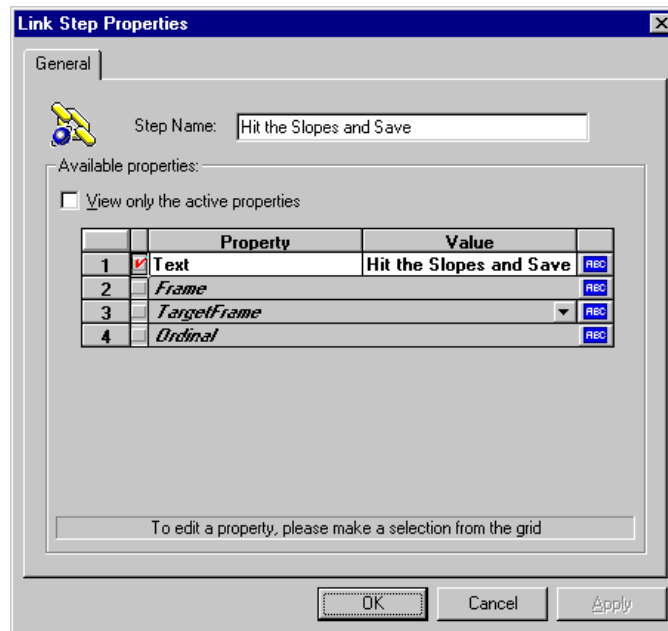
A hypertext link step is added to the Web Vuser script when you click a hypertext link. This step is only recorded when you select the option to record in **HTML based script** mode. For more information, refer to Chapter 35, “Setting Recording Options for Web Vusers.”

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

► **Ordinal:** a number that uniquely identifies the link when all the other property attributes are identical to one or more other links on the Web page. See the *Online Function Reference* for details.

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see Chapter 7, “Defining Parameters.”

5 Click **OK** to close the Link Step Properties dialog box.

Modifying an Image Step (Web only)

An image step is added to the Vuser script when you click a hypergraphic link. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. For more information, refer to Chapter 35, “Setting Recording Options for Web Vusers.”

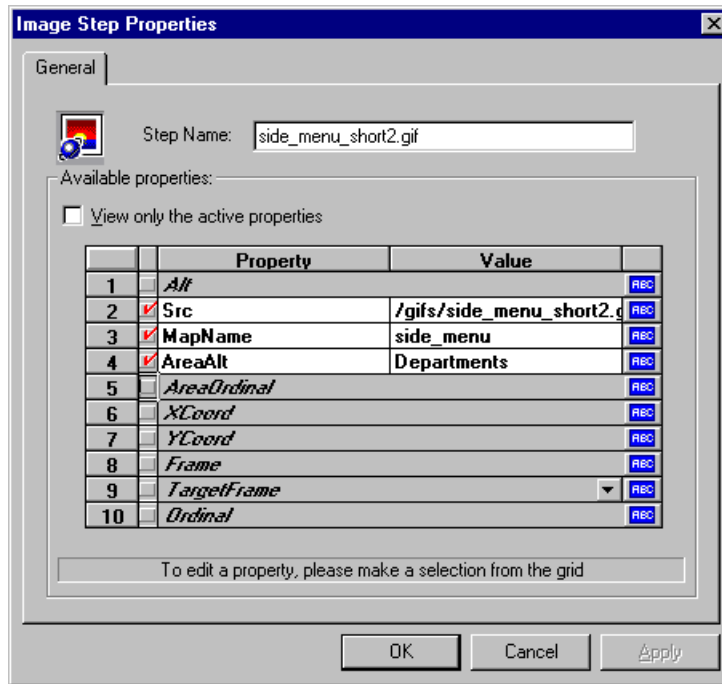
The properties that you can modify are the name of the step, how the hypergraphic link is identified, and where it is located.

To modify the properties of an image step:



1 In the tree view of the Vuser script, select the image step you want to edit. Image steps are shown using the Image icon.

- 2 Select **Properties** from the right-click menu. The Image Step Properties dialog box opens.



- 3 To change the step name, type a new name in the **Step Name** box. The default name during recording is the image's ALT attribute. If the image does not have an ALT attribute, then the last part of the SRC attribute is used as the default name.

- 4 The properties table displays the properties that identify the link.

Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **ALT:** the ALT attribute of the image
- **SRC:** the SRC attribute of the image
- **MapName:** the name of the map related to the image. Applies to client-side image maps only.

- **AreaAlt:** the ALT attribute of the area to click. Applies to client-side image maps only.
- **AreaOrdinal:** the serial number of the area to click. Applies to client-side image maps only.
- **Frame:** the name of the frame where the image is located
- **TargetFrame:** the target frame:
 - _TOP: replaces the whole page
 - _BLANK: opens a new window
 - _PARENT: replaces the parent of the last (changed) frame
 - _SELF: replaces the last (changed) frame
- **Ordinal:** a number that uniquely identifies the image when all other property attributes are identical to one or more other images on the Web page. See the *Online Function Reference* for details.
- **XCoord, YCoord:** the coordinates of the mouse-click on the image

An ABC icon indicates that the link property value has not been assigned a parameter. For details on assigning parameters, see Chapter 7, “Defining Parameters.”

- 5 Click **OK** to close the Image Step Properties dialog box.

Modifying a Submit Form Step (Web only)

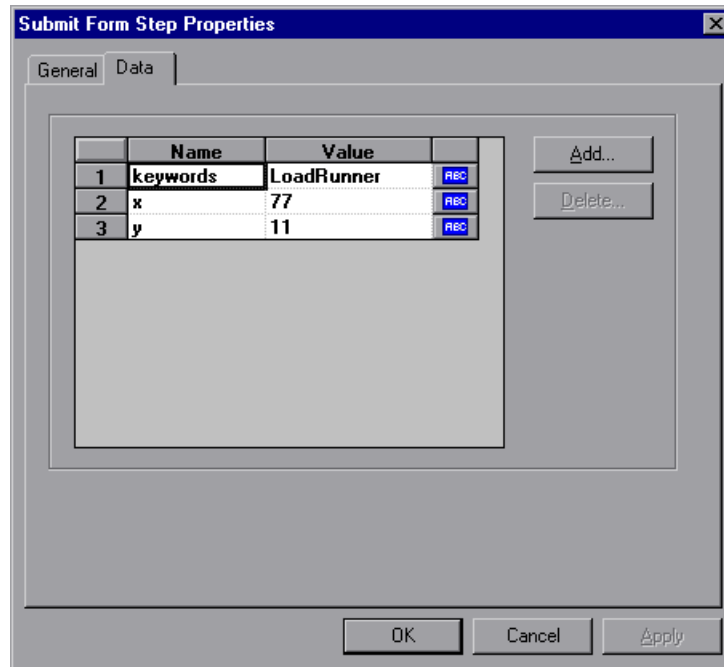
A submit form step is added to the Vuser script when you submit a form. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. For more information, refer to Chapter 35, “Setting Recording Options for Web Vusers.”

The properties that you can modify are the name of the step, the form location, how the form submission is identified, and the form data.

To modify the properties of a submit form step:

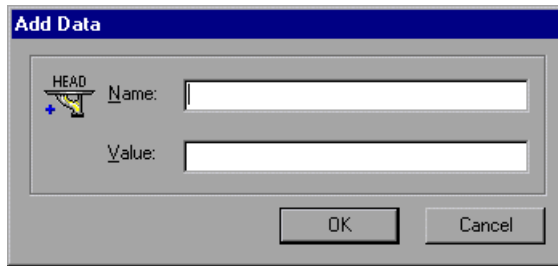
- 1 In the tree view of the Vuser script, select the submit form step you want to edit. Submit form steps are shown using the Submit Form icon.

- 2 Select **Properties** from the right-click menu. The Submit Form Step Properties dialog box opens. Ensure that the **Data** tab is selected.

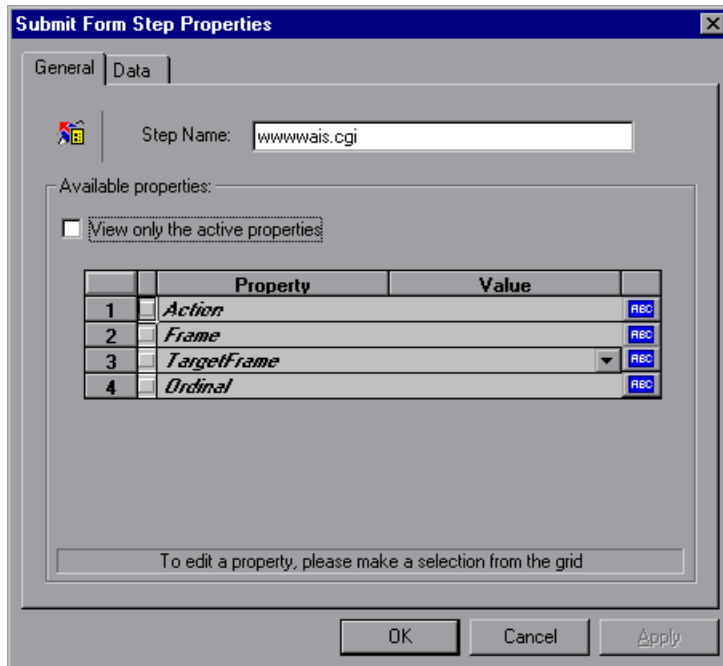


- The **Name** column lists all the data arguments on the form.
 - The **Value** column displays the corresponding value input for a data argument.
 - The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 7, “Defining Parameters,” the ABC icon changes to a table icon.
- 3 To edit a data argument, double-click on it to activate the cursor within the cell. Then type the new value.

- 4 To add a new data argument to the form submission, click **Add**. The Add Data dialog box opens.



- 5 Type a **Name** and **Value** for the data argument, and click **OK**.
- 6 To delete an argument, select it and click **Delete**.
- 7 To change the name of the submit form step, click the **General** tab.



- 8 To change the step name, type a new name in the **Step Name** box. The default name during recording is the name of the executable program used to process the form.
- 9 The properties table displays the properties that identify the form submission.

Clear the **View only the active properties** option to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- ▶ **Action:** the address to be used to carry out the action of the form
- ▶ **Frame:** the name of the frame where the form submission is located
- ▶ **TargetFrame:** the target frame:
 - _TOP: replaces the whole page
 - _BLANK: opens a new window
 - _PARENT: replaces the parent of the last (changed) frame
 - _SELF: replaces the last (changed) frame
- ▶ **Ordinal:** a number that uniquely identifies the form when all other property attributes are identical to one or more other forms on the same Web page. See the *Online Function Reference* for details. (**Help > Function Reference**)

An ABC icon indicates that the submit form step property value has not been assigned a parameter. For details on assigning parameters, see Chapter 7, “Defining Parameters.”

- 10 Click **OK** to close the Submit Form Step Properties dialog box.

Modifying a Submit Data Step

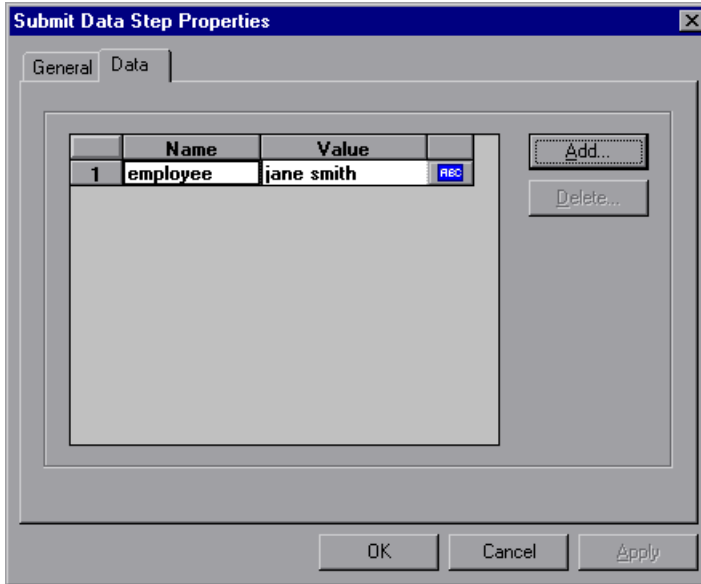
A submit data step represents the submission of a form of data to your Web site for processing. This is different from a Submit Form step because you do not need to have a form context to execute this request.

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:

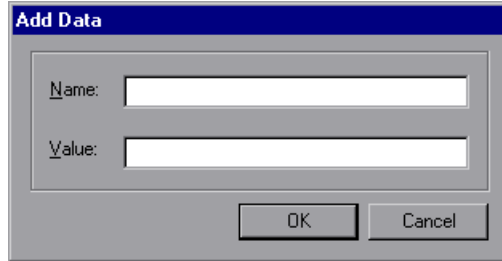


- 1** In the tree view of the Vuser script, select the submit data step you want to edit. Submit data steps are shown using the Submit Data icon.
- 2** Select **Properties** from the right-click menu. The Submit Data Step Properties dialog box opens. Ensure that **Data** tab is visible.

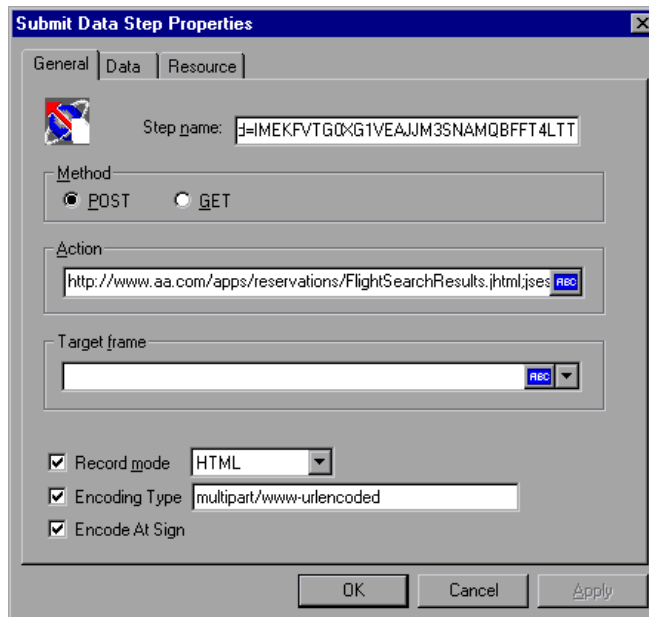


- The **Name** column lists all the data arguments on the form. This includes all hidden fields.
 - The **Value** column displays the corresponding value input for a data argument.
 - The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 7, “Defining Parameters,” the ABC icon changes to a table icon.
- 3** To edit a data argument, double-click on it to activate the cursor within the cell. Then type the new value.

- 4 To add new data, click **Add**. The Add Data dialog box opens.



- 5 Type a **Name** and **Value** for the data argument, and click **OK**.
- 6 To delete an argument, select it and click **Delete**.
- 7 To change the name of the submit data step, click the **General** tab.



- 8 To change the step name, type a new name in the **Step name** box.
- 9 Under **Method**, click **POST** or **GET**. The default method is **POST**.
- 10 In the **Action** box, type the address to be used to carry out the action of the data submission. An ABC icon indicates that the action has not been

assigned a parameter. For details on assigning parameters, see Chapter 7, “Defining Parameters.”

- 11** Select a **Target frame** from the list:
 - _TOP: replaces the whole page
 - _BLANK: opens a new window
 - _PARENT: replaces the parent of the last (changed) frame
 - _SELF: replaces the last (changed) frame
- 12** To customize the replay mode, select the **Record mode** option. Choose the desired mode: HTML, or HTTP. For more information, see “Setting the Replay Mode,” on page 497.
- 13** To exclude an item from being downloaded as a resource, clear the **Resource** check box.
- 14** To specify an encoding type, such as *multipart/www-urlencoded*, select the **Encoding type** check box and specify the encoding method.
- 15** To encode the “@” in the URL, select **Encode At sign**.
- 16** Click **OK** to close the Submit Data Step Properties dialog box.

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.

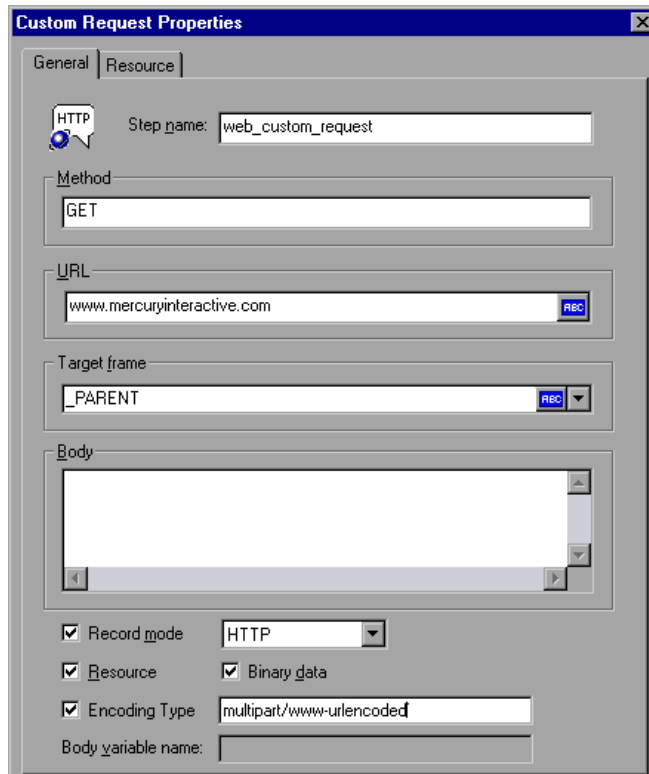
VuGen has a feature that lets you convert a custom request body string to C format. For example, if you copy an XML tree or a large amount of data into the Body area of the custom request, you can easily convert the strings to C format in order that it may be incorporated into the current function. It 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** box, type the body of the request or past 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**)
- 8** For strings that you pasted into the **Body** box, select the text and choose **Convert to C format** from the right-click menu.
- 9** To customize the replay mode, select the **Record mode** option. Choose the desired mode: HTML, or HTTP. For more information, see “Setting the Replay Mode,” on page 497.
- 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 control used during tuning. 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

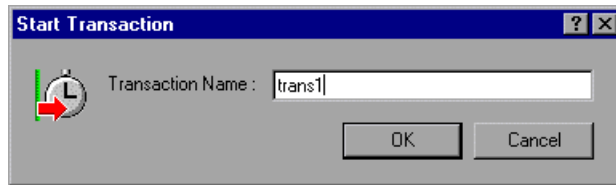
Modifying a Transaction

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:

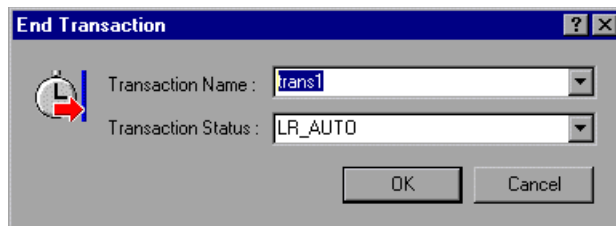
- 1 In the tree view of the Vuser script, select the start transaction control step you want to edit. Start transaction control steps are shown using the Start Transaction icon.
- 2 Select **Properties** from the right-click menu. The Start Transaction dialog box opens.



- 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

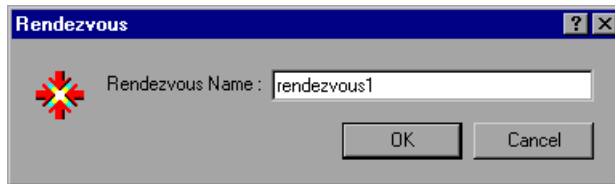
Rendezvous points enable you to synchronize Vusers to perform a task at exactly the same moment.

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**.

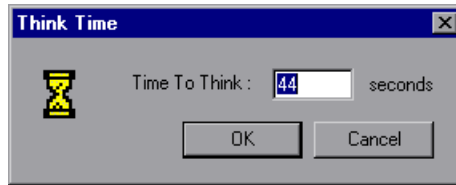
Modifying Think Time

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 a predefined threshold of about four seconds.

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, either in VuGen or from the Console, you can instruct the Vuser to replay think time as recorded or ignore the recorded think time. For details, see Chapter 9, “Configuring Run-Time Settings.”

Modifying Service Steps

A service step is a function that performs customization tasks such as setting proxies, submitting authorization information, and issuing customized headers. Service steps do not make any changes to the Web site context.

You add service steps to your script during and after recording.

To modify the properties of a service step:

- 1 In the tree view of the Vuser script, select the service step you want to edit. Service steps are shown using the Service icon.
- 2 Select **Properties** from the right-click menu. The appropriate service step properties dialog box opens. This dialog box varies, depending on the type of service step that you are modifying. A description of the service step is displayed in the title bar of the dialog box.

Note: Some service step functions have no arguments. In these cases, the Properties menu item is disabled.

- 3 Type or select the arguments required for the service step. See the *Online Function Reference* for details of each function. (**Help > Function Reference**)
- 4 Click **OK** to close the service step properties dialog box.

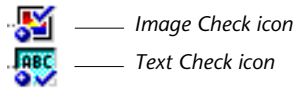
Modifying Web Checks (Web only)

A Web check is a function that verifies the presence of a specific object on a Web page. The object can be a text string or an image.

You add Web checks to your script during and after recording.

To modify the properties of a Web check:

- 1 In the tree view of the Vuser script, select the Web check you want to edit. Web checks are shown using Web Check icons.



- 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 38, “Verifying Web Pages Under Load.”
- 4 Click **OK** to close the check properties dialog box.

40

Configuring Correlation Rules for Web Vuser Scripts

VuGen's correlation feature allows you to link Vuser functions by using the results of one statement as input to another.

This chapter describes how to correlate statements during recording. It discusses:

- ▶ Understanding the Correlation Methods
- ▶ Choosing a Correlation Handling Method
- ▶ Testing Rules
- ▶ Setting the Correlation Recording Options

The following information applies to Web Vuser scripts.

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 variables 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. In many cases, correlation will solve the

problem by enabling you to use the results of one statement as input to 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
- a field (sometimes hidden) recorded during a form submission
- 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=170397&ID=1234"
```

Since the date "170397" 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, ProTune is 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, ProTune assigns parameter values.

Understanding the Correlation Methods

This section discusses automatic correlation, using built-in or user-defined rules. To manually correlate statements, or to perform correlation for Wireless Vuser scripts, refer to “Performing Manual Correlation,” on page 539.

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 “Selecting a Recording Mode,” on page 445.

You can instruct ProTune 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, see Chapter 41, “Correlating Vuser Scripts After Recording.”

VuGen’s correlation engine allows you to automatically correlate dynamic data during your recording session using one of these mechanisms:

- Built-In Correlation
- User-Defined Rule Correlation

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 41, “Correlating Vuser Scripts After Recording.”

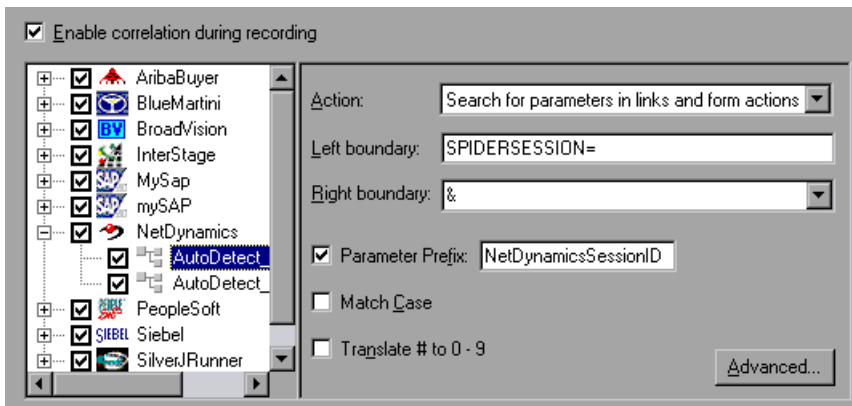
Built-In Correlation

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-in to 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.



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

All Body Text

The **Search for Parameters in all of the Body Text** option instructs the recorder to search the entire body for a match—not just links, form actions or cookies. It searches for text, 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 form type 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, choose *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 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 pull-down list. If none of the right boundaries are found, the text from the left boundary to the end of source string is saved to the parameter.

Note that for this option, the left and right boundaries must appear in the string which 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.

In addition, the link/ form action rule parameterizes only the part of URL which matches the rule boundaries. The cookie rule looks for the extracted value in links and action form fields and replaces it with a parameter without having to specify any boundaries.

Form Field Value

The **Parameterize form field value** method instructs the recorder to save all form fields that have names to a parameter. It creates a parameter and places it in the script before the form's action step. This method is ideal for recording sites with many forms. For this options, you need to specify either the left and right boundaries or the field name (for the Form Field option).

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 more easily.

Match Case: Matches the case when looking for boundaries.

Use “#” for any digit: Replaces all digits with a hash sign. This will allow you to find text strings that match everything except for numerical digits. For example, if the left boundary is *Mercury193*, it will match *Mercury284*. In the left boundary box, specify *Mercury###*.

VuGen also lets you specify several advanced options:

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*. The Name argument of the `web_submit_data` is `Name=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 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) in order 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 of a sub-string of the matched string, to save to the parameter, from its offset. If you disable this option, it uses the default and 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.

Choosing a Correlation Handling Method

When you enable correlation, checks to see if the rule exists for your application. If the dynamic data conforms to an existing rule, VuGen prepares to perform correlation, based on the following settings:

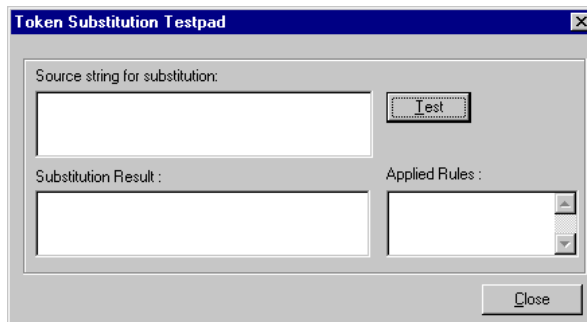
- ▶ **Issue a popup message and let me decide online:** Issue a message during the recording when detecting dynamic data, before performing correlation.
- ▶ **Perform correlation in script:** Automatically correlate the statement within the script.

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.

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**.

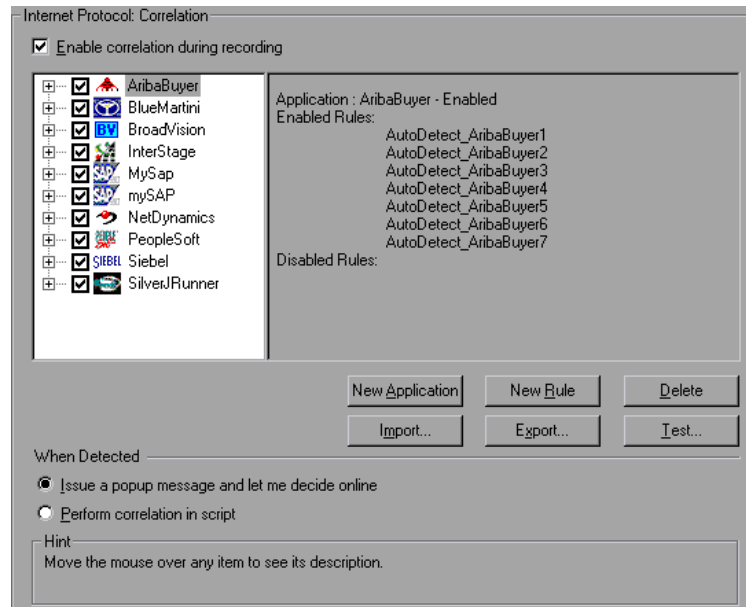
If substitution occurred, you will see the parameterized source text in the **Substitution Result** box, and a list of rules that were applied to it in the **Applied Rules** box.

Setting the Correlation Recording Options

To instruct ProTune to correlate your statements during recording, you set the Correlation recording options. You set these options after opening a Web Vuser script but before you begin recording the session.

To set the correlation recording options:

- 1 After you create a script, but before you begin recording, select **Tools > Recording Options** and select the **Internet Protocol:Correlation** node in the Recording Options tree.



- 2 Select the **Enable correlation during recording** option.
- 3 Indicate the servers to which you want to apply the correlation rules. Select the check boxes adjacent to the server names to enable the rules for that

server. To enable specific rules within a server group, click the plus sign to expand the tree and select the desired rules.

- 4 To add a new rule to an existing server, select one of the existing entries and click **New Rule**. Set the properties for the rule in the right pane. For more information, see “Setting Correlation Rules,” on page 524.
- 5 To add a set of rules for a new application, click **New Application**. Then click **New Rule** to create a rule for the application.
- 6 To modify the properties of an existing rule, select the rule in the left pane and modify the rules in the right pane.
- 7 Indicate what VuGen should do when it detects a value that needs to be correlated: **Issue a popup message** or **Perform correlation in script**. By default, ProTune issues a popup message.
- 8 To delete an application or rule, select it and click **Delete**. VuGen prompts you with a warning before deleting the selection.
- 9 To export a set of correlation rules, click **Export** and save the *cor* file to the desired location. To import a set of correlation rules created during an earlier session, click **Import** and open the file from its location.
- 10 Click **OK**.

Setting Correlation Rules

You can add, modify, or remove rules using the Correlation options. Note that you can also edit rules that were created automatically for application server environments.

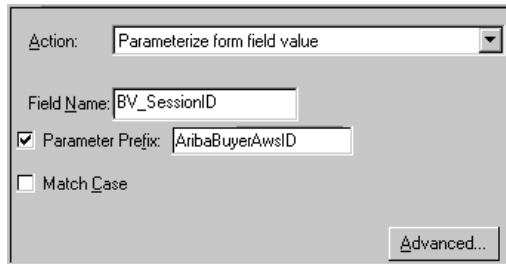
To define correlation rules:

- 1 Click on an existing rule or click **New Rule** in the left pane. The Correlation Rules are displayed in the right pane.

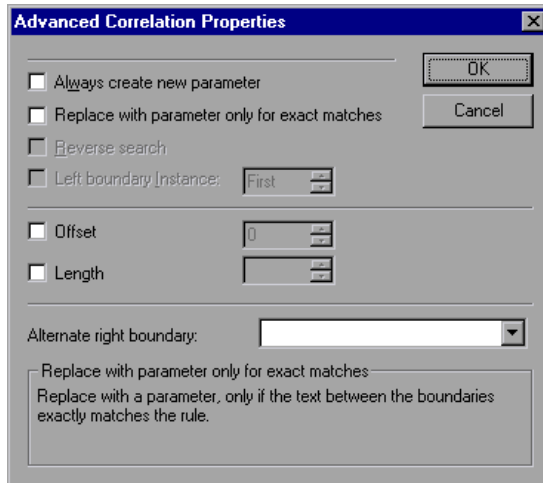
The screenshot shows the configuration window for a correlation rule. It includes the following fields and options:

- Action:** A dropdown menu set to "Search for parameters in links and form actions".
- Left boundary:** A text input field containing "aws=".
- Right boundary:** A dropdown menu set to "&".
- Parameter Prefix:** A checked checkbox followed by a text input field containing "AribaBuyerAwsID".
- Match Case:** An unchecked checkbox.

- 2 Select a type of action: link or form action, cookie, all body, or form field.
- 3 For the first three types, specify boundaries of the data in **Left Boundary** and **Right Boundary** boxes.
- 4 For form field type actions, specify the field name.



- 5 Select the desired options: **Match Case** and/or **Parameter Prefix**. Specify a parameter prefix. To convert all digits to hash signs (#), select **Use # for any digit**.
- 6 To set advanced rules, click **Advanced**. 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 choose **End of String** or **NewLine Character** from the pull-down list.
- 7** Click **Test Rule** to test the rule you just defined. For information, see “Testing Rules,” on page 522.
- 8** Click **OK** to save the rules and close the dialog box.

41

Correlating Vuser Scripts 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 describes:

- ▶ Understanding Snapshots
- ▶ Setting Up VuGen for Correlation
- ▶ Performing a Scan for Correlations
- ▶ Performing Manual Correlation
- ▶ Defining a Dynamic String's Boundaries

The following information only applies to Web and Wireless Vuser scripts.

About Correlating with Snapshots

VuGen provides several correlation mechanisms for Web Vuser scripts. The automatic method discussed in Chapter 40, "Configuring Correlation Rules for Web Vuser Scripts", detects dynamic values during recording and allows you to correlate them immediately. 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 creates a base snapshot during recording, and generates a new snapshot

every time you execute the script. You compare the recorded snapshot to any one of the replay snapshots to determine which values you need to correlate in order to insure a successful execution of the script.

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 539.

Understanding Snapshots

The correlation mechanism uses *snapshots* to track the results of script execution. VuGen creates a base snapshot during recording, and generates a new snapshot every time you execute the script. You compare the snapshots and their HTML code to find the dynamic values that need to be correlated in order to run the script.

The snapshot files are stored under the script directory with an *.inf* extension. Snapshots created during recording are stored in the Vuser script's *data* folder. The Replay snapshots are located in the script's Iteration folders: *Iteration1*, *Iteration2*, etc. for each set of results. By default, VuGen compares the Recording snapshot to the first replay snapshot. You may, however, choose a different snapshot for comparison.

If there is no recording snapshot displayed for the selected step, check the following possible reasons:

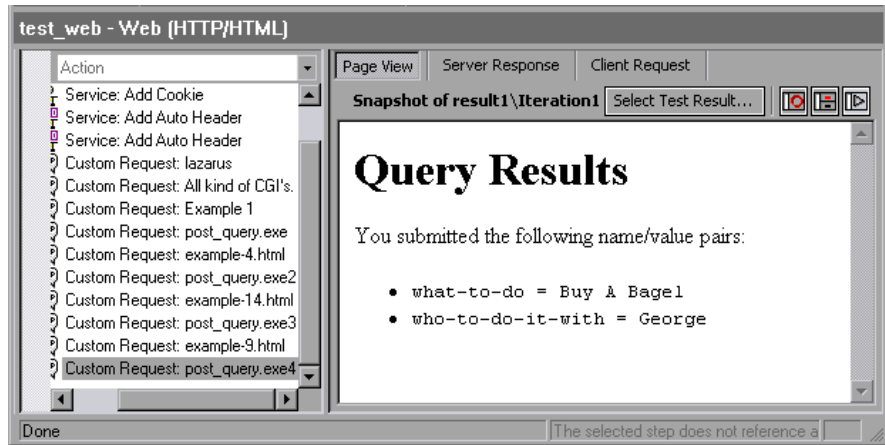
- Snapshots are not generated for certain types of steps.
- The imported actions do not contain snapshots.

If there is no replay snapshot displayed for the selected step, check the following possible reasons:

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

- The step represents navigation to a resource.
- The following option was turned off to disable snapshot generation:
Tools > General options > Correlation tab > Save correlation information during replay.

By default, when working in Tree view, VuGen does not display the snapshots of the selected step in the right pane. To hide or show the snapshots, choose **View > Snapshot**.



Use the expanded menu of **View > Snapshot** to view the Recorded, Replayed, or both snapshots. You can also use shortcut buttons to display the desired view:



Show a full window of the Recorded snapshot



Show a split window of the Recorded and Replayed snapshot



Show a full window of the Replayed snapshot

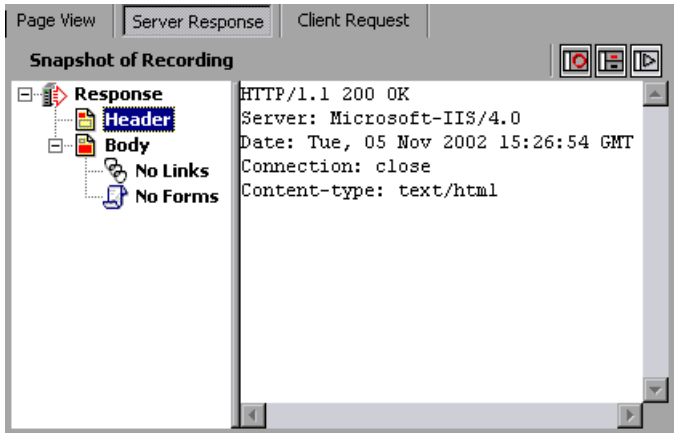
To determine the name of the snapshot file, view the script in Script view (**View > Script View**). In the following example, the snapshot information is represented by *t1.inf*.

```
web_url("www.aa.com",
        "URL=http://www.aa.com/",
        "Resource=0",
        "RecContentType=text/html",
        "SupportFrames=0",
        "Snapshot=t1.inf",
        LAST);
```

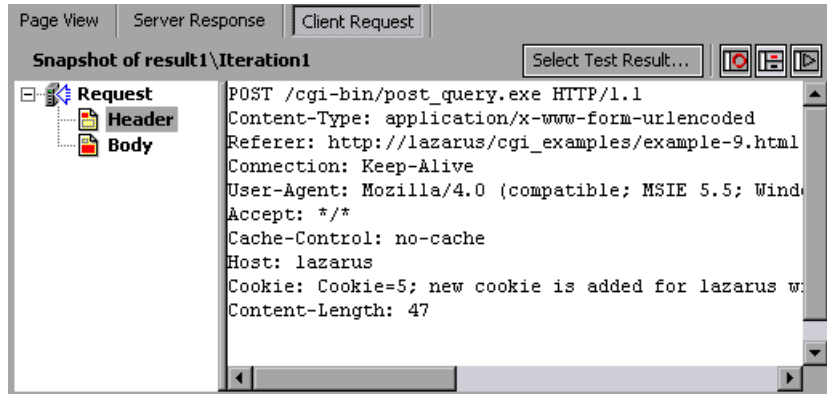
In the Snapshot windows, the following tabs are available:

Page View: Display the snapshot in HTML as it would appear in a browser. This button is available for both the Recording and Replay snapshots. Use this view to make sure you are viewing the correct snapshot. In this view, however, you do not see the values that need to be correlated.

Server Response: Displays the server response HTML code of the snapshot. This button is available for both the Recorded and Replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body with the title, links, forms, etc.



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



Select Test Result: (for Replayed snapshots only) Lets you select a Replayed snapshot to compare with the recording snapshot. This button opens a dialog box which lists the results folders for the active script. The Replayed snapshots are located in the script's Iteration folders: *Iteration1*, *Iteration2*, etc. for each set of results.

When you instruct VuGen to scan the script for correlations, it opens the Output window and displays the **Correlation Results** tab. In the **Correlation Results** tab it displays the differences between the Record and Replay snapshots.

Show Differences In: Entire Script

Text in Recording Snapshot	Text in result1\Iteration1	Count
http:	https:	1
www.delta.com	security	1
home	index.jsp	1
cgi-bin	components	1
delta	skymiles_login.jsp	1
6	12	1
index.jsp	home	1

Correlate
Correlate All
Undo
Undo All

You can display all the differences in the script or only those for the current step by selecting the desired option from the **Show Differences In** list box.

Differences that were correlated are indicated by a check mark in the leftmost column. The next two columns show the HTML differences between the snapshots. The rightmost column, *count*, indicates the number of occurrences of that difference between the recorded snapshots.

Show Differences In: Entire Script

Text in Recording Snapshot	Text in result1\Iteration1	Count
http:	https:	1
✓ www.delta.com	security	1
home	index.jsp	1
cgi-bin	components	1
delta	skymiles_login.jsp	1
6	12	1

Buttons: Correlate, Correlate All, Undo, Undo All

After you detect the differences between the snapshots, you correlate them either one at a time (**Correlate**), or all at once (**Correlate All**). VuGen also allows you to undo a specific correlation (**Undo**) or all correlations (**Undo All**).

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.

```
// Correlation Studio created parameter {WCSParam_Diff1}; replaced
value:falillgidgkbfldlclmcfkkgdggff.0
web_reg_save_param( "WCSParam_Diff1", "LB=BV_EngineID=", "RB=&",
"Ord=1", "Search=body", LAST );
web_url("American2",
        "URL=http://www.im.aa.com/Ameri-
can?BV_EngineID={WCSParam_Diff1}&BV_Operation=Dyn_Frame&form
%25framespacing=0&BV_SessionID=%40%40%40%401303778278.096
9956817%40%40%40%40&form%25destination=%2fnavguest.tmpl&fo
rm%25destination_type=tem-
plate&form%25border=0&BV_ServiceName=Ameri-
can&form%25frameborder=no",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "SupportFrames=0",
        "Referer=http://www.im.aa.com/Ameri
can?BV_Operation=Dyn_AAPage&ref
erer=index.html&form%25referrer_site=None",
        "Snapshot=t3.inf",
        LAST);
```

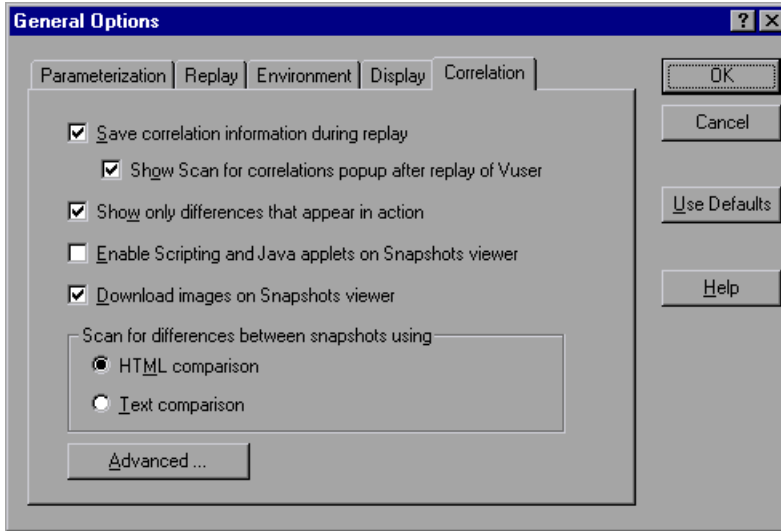
Setting Up VuGen for Correlation

Before Recording you set up the general correlation options. These options instruct the Vusers to save correlation information during recording, to be used at a later stage. You can specify the type of comparison to perform when comparing snapshots: HTML or text. In the **Correlation** tab, you can also indicate which characters should be treated as delimiters.

Note: In most cases, it is recommended that you work with the default HTML comparison method. If your script contains non-HTML tags, you can use the Text comparison method.

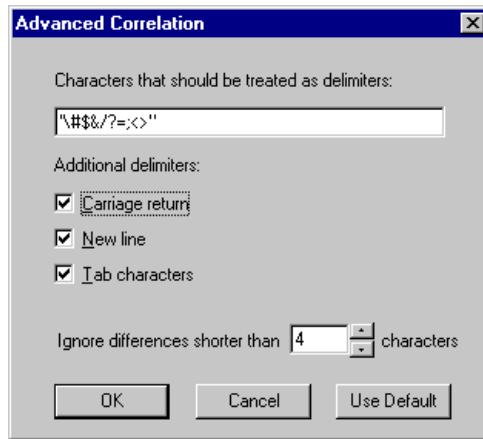
To set the correlation preferences:

- 1 To open the correlation options dialog box, choose **Options > General** and select the **Correlation** tab.



- 2 To save the replay information as snapshots, select the **Save correlation information during replay** check box. You can compare any of the replay snapshots to the recording snapshot.
- 3 To instruct VuGen to prompt you before scanning the script for correlations, select the **Show Scan for correlations popup after replay of Vuser**. VuGen prompts you after replay, before scanning the script.
- 4 The statements that appear in the current action are the statements that generate calls to HTML pages but not the HTML data returned from server. In most cases this information is enough for correlation. Any dynamic data that is later used in a statement and requires correlation appears in the current action. To display the differences that appear in the current action, choose **Show only differences that appear in action**. In rare cases where you want to create a parameter from data that does not appear in the action, disable this option.
- 5 Select **Enable Scripting and Java applets on Snapshots viewer** to allow VuGen to run applets and javascript in the snapshot window. This is disabled by default since it requires more resources.

- 6 To instruct VuGen to display graphics in the Snapshot view, select the **Download images on Snapshots viewer** option. You can disable this option if, for example, you know that no graphics require correlation. Disabling the option saves resources. This option is enabled by default.
- 7 Choose the comparison method: **HTML comparison** or **Text Comparison** (for non-HTML elements only).
- 8 To set the delimiter characters, click **Advanced** to open the Advanced Correlation dialog box.



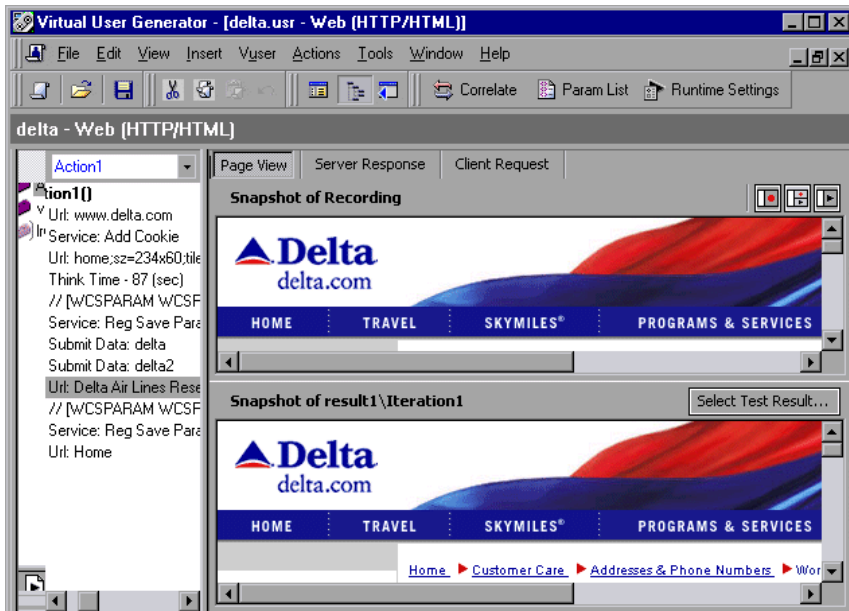
- 9 Specify all characters that are to be treated as delimiters.
- 10 Select the desired check boxes in the **Additional delimiters** section, to specify one or more standard delimiters.
- 11 Specify a threshold for the correlation in the **Ignore differences shorter than** box. When VuGen compares the recorded script with the executed script 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.
- 12 Click **OK** to accept the settings and close the dialog box.
- 13 Click **OK** in the General Options dialog box to accept the Correlation setting and close the dialog box.

Performing a Scan for Correlations

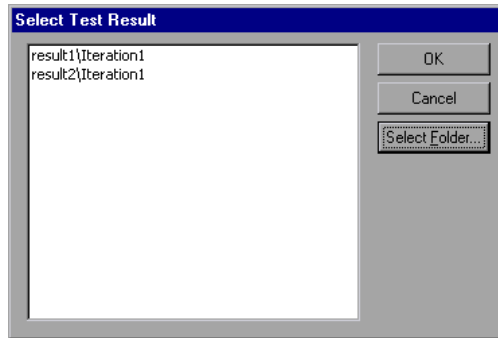
You can use VuGen's snapshot window to determine which values within your script are dynamic and require correlation. The following section describes how to automatically scan the script for differences and use VuGen to perform the necessary correlations.

To scan your script for correlations:

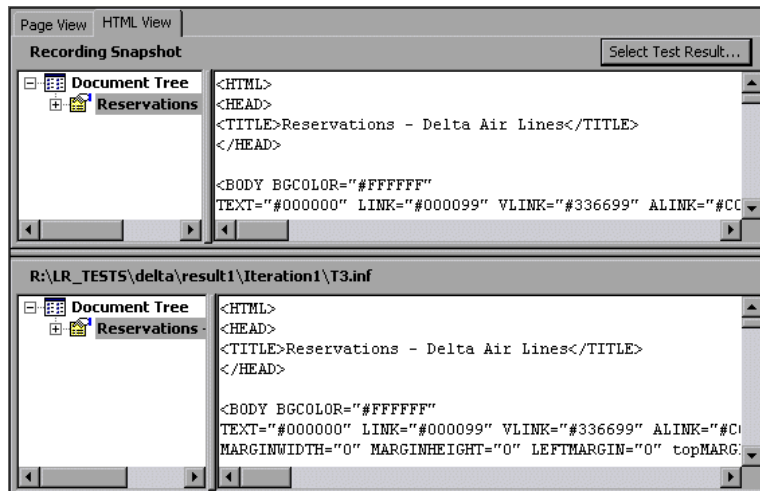
- 1** Open a script and view it in Tree view (**View > Tree View**). Display the snapshots (**View > Snapshot**).
- 2** Select a script step in the Tree view from the left pane. The Recording snapshot and the first replay snapshot open in the right pane.



- 3 To use a snapshot other than the first, click **Select Test Result**. A dialog box opens, displaying the folders that contain snapshot files. These are usually the *result* and *Iteration* folders below the script's folder.



- 4 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**.
- 5 To view the HTML code, click the **Server Request** tab. Expand the Body branch. To return to the page view, click the **Page View** tab.





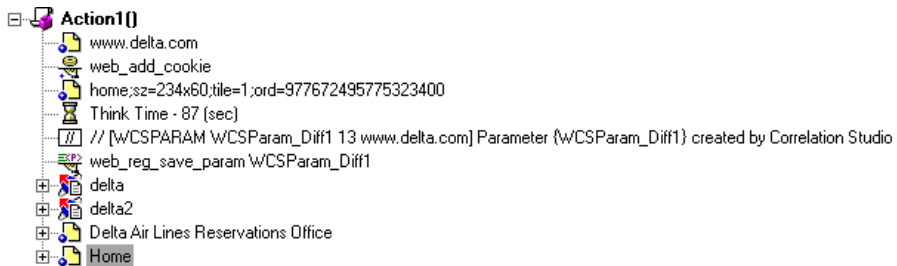
- 6 Choose **Vuser > Scan for Correlations** or click the **Scan for Correlations** button. VuGen scans the script for dynamic values that need to be correlated and displays them in the Correlation Results window.

Show Differences In: Entire Script

Text in Recording Snapshot	Text in result1\Iteration1	Count
http:	https:	1
www.delta.com	security	1
home	index.jsp	1
cgi-bin	components	1
delta	skymiles_login.jsp	1
6	12	1
index.jsp	home	1

Buttons: Correlate, Correlate All, Undo, Undo All

- 7 To view the differences in a specific step of the Vuser script, select the step in VuGen's tree view and select the **Current Step** in the **Show Differences In** list box. To view all differences, choose **Entire Script** in the **Show Differences In** list box.
- 8 To correlate all differences, click **Correlate All**. To correlate a specific difference, select it and click **Correlate**. VuGen places a green check mark next to differences that were correlated and inserts **web_reg_save_param** functions into the Vuser script.



- 9 To undo a correlation, select the difference and click **Undo**. To undo all correlations, click **Undo All**.
- 10 Choose **File > Save** to save the changes.

Performing Manual Correlation

For Web Vusers, VuGen's automatic or rule-based correlation usually correlates the script's dynamic functions so that you can run the script successfully. You can also perform correlation after the recording session using VuGen's snapshot comparison. (See Chapter 41, "Correlating Vuser Scripts After Recording.")

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 correlation 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 (e.g. Search=all) are not case sensitive.

<i>NotFound</i>	The handling method when a boundary is not found and an empty string is generated. "ERROR", the default, indicates that ProTune 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.
<i>LB</i>	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.
<i>RB</i>	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.
<i>RelFrameID</i>	The hierarchy level of the HTML page relative to the requested URL. The possible values are ALL or a number.
<i>Search</i>	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.
<i>ORD</i>	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.

<i>SaveOffset</i>	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.
<i>SaveLen</i>	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.
<i>Convert</i>	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 544.
- 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 542 or the *Online Function Reference*. (**Help > Function Reference**)

Replacing Dynamic Data with a Parameter

First, identify the actual dynamic data. After you locate the dynamic data in the recorded statement, 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 run-time 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 and 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 Vusers

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 border, 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");
```


42

Testing XML Pages

VuGen's Web Vusers support Web pages containing XML code.

This chapter describes:

- ▶ Viewing XML as URL steps
- ▶ Inserting XML as a Custom Request
- ▶ Viewing XML Custom Request Steps

The following information only applies to Web Vuser scripts.

About Testing XML Pages

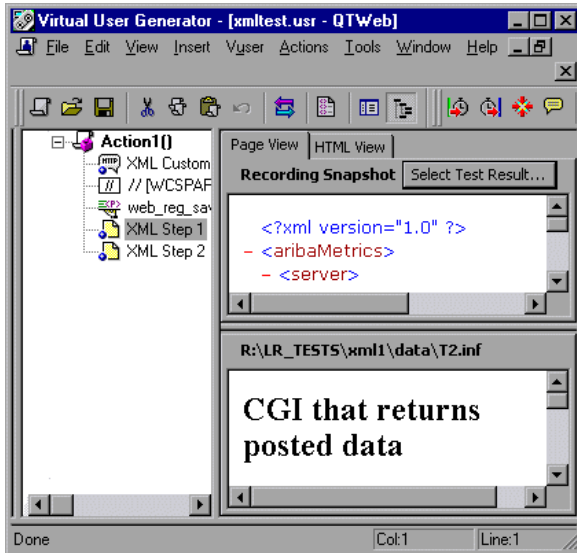
VuGen supports record and replay for XML code within Web pages.

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. The DTD is color coded, allowing you to easily identify the elements. You can also expand and collapse the tree view of the DTD. Note that VuGen can detect the XML even when the MIME type displayed in the **RecContentType** attribute is not set to *text/xml*.

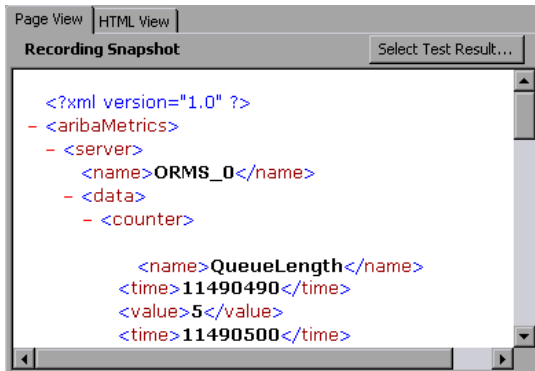
When you expand the DTD, you can parameterize the attribute values. You can also save the values in order to perform correlation using the standard correlation functions. For more information about the correlation functions, see the *Online Function Reference*. (**Help > Function Reference**)

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.



VuGen creates a color-coded expandable hierarchy of the DTD in the snapshot frame. Click on the "+" to expand an item, and click on the "-" to collapse it. VuGen displays all XML tags in brown, and values in black.



To replace any of the constant values with a parameter, select a value, perform a right-click, and select **Replace with a Parameter**. Follow the standard procedure for parameterization. For more information, see the chapter on parameterization.

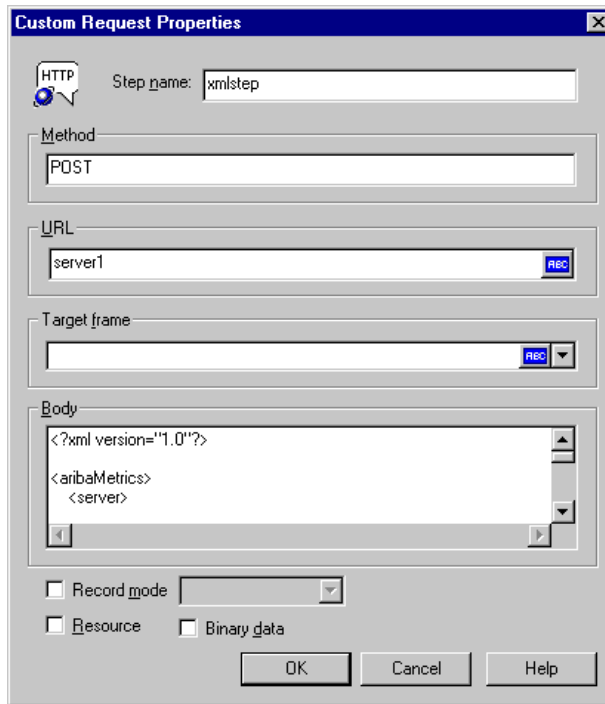
Inserting XML as a Custom Request

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 choose **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 39, “Modifying Web and Wireless Vuser Scripts”.
- 6 Click **OK**. VuGen places the custom request step into your script.

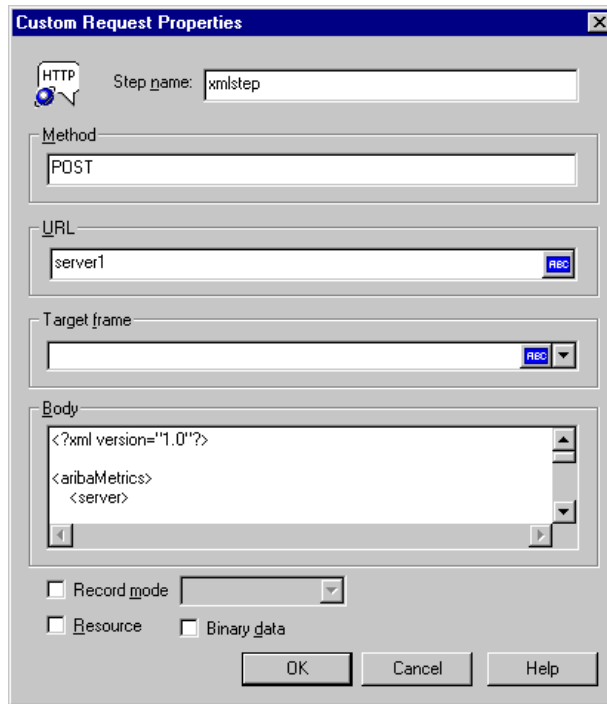
Viewing XML Custom Request Steps

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.

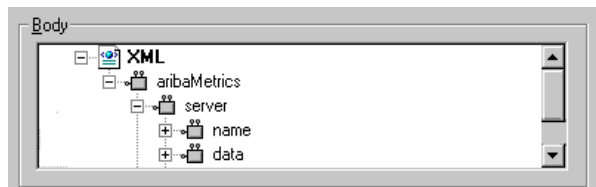
To view the XML code of a custom request step:

- 1 View the script in tree view mode, and select the desired step.

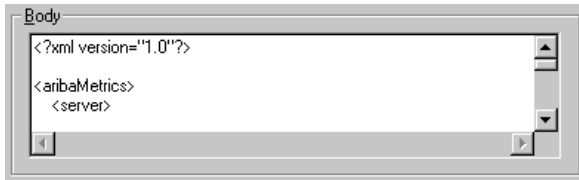
- 2 Choose **Properties** from the right-click menu. The Custom Request Properties dialog box appears.



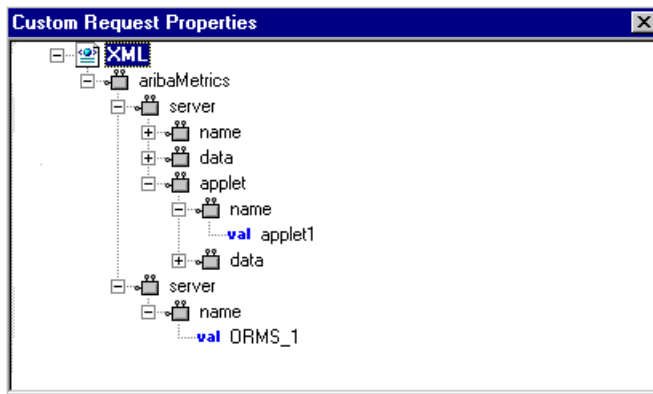
The bottom section of the dialog box displays the XML code. If the **RecContentType** attribute is set to *text/xml*, by default VuGen displays the code in an XML format hierarchy. In this mode, the XML code is not editable.



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, choose **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. Choose **Extended view** from the right-click menu. To switch back to the dialog box view, choose **Normal view** from the right-click menu.



43

Using Reports to Debug Vuser Scripts

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

This chapter describes:

- Understanding the Results Summary Report
- Filtering Report Information
- Managing Execution Results

Note: To enable all the VuGen Web report features, it is recommended that you work with Microsoft Internet Explorer 4.0 or higher.

The following information only applies to Web Vuser scripts.

About Using Reports to Debug Vuser Scripts

When you debug a Web Vuser script using VuGen, you specify whether or not to generate a *Results Summary* report during script execution. The Results Summary report contains details of all the Web pages that the Vuser visited as well as any checks that the Vuser performed. Examining this information is useful when debugging the Web Vuser script. For details on running Vuser scripts using VuGen, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

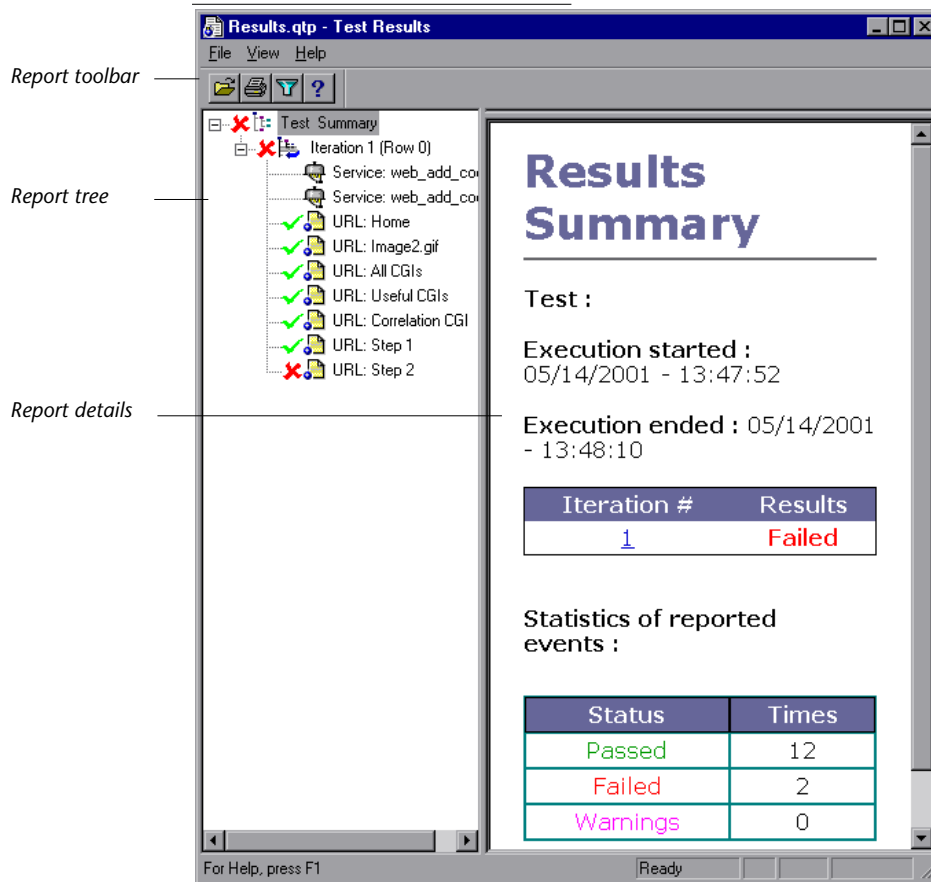
After you run a Vuser script using VuGen, you view the Results Summary report.

If Microsoft Internet Explorer 4.0 or higher is installed on your machine, VuGen generates the results in VuGen report format—with a *.qtp* extension—and you view the results in the Virtual User Generator Report window. This is the recommended option because VuGen's Report window provides you with a more sophisticated interface and additional features.

You set the Visual Log options (**Tools > General Options**) to specify whether or not VuGen should generate a Results Summary report, and if so, whether the report opens automatically after script execution. For details on setting the Visual Log options, see Chapter 10, "Running Vuser Scripts in Stand-Alone Mode."





Understanding the Results Summary Report

After running your Vuser script, you view the Results Summary report. The report displays a summary of the results of the script execution.



- The left pane displays the *report tree*—a graphical representation of the results. In the report tree, a green check mark represents a successful step, and a red X represents a failed step.
- The right pane displays the *report details*—an overall summary of the script run, as well as additional information for a selected branch of the report tree.

You select a branch of the report tree to view the information for that branch.

Select the branch...	To view the following details:
Test Name  Test mercury	the overall results summary of the script execution
Test Iteration  Test Iteration	the execution summary for a specific iteration
Test Step or Check  Link: Contact Us  Text Check:	the Web page for the selected step or check in the Vuser script

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

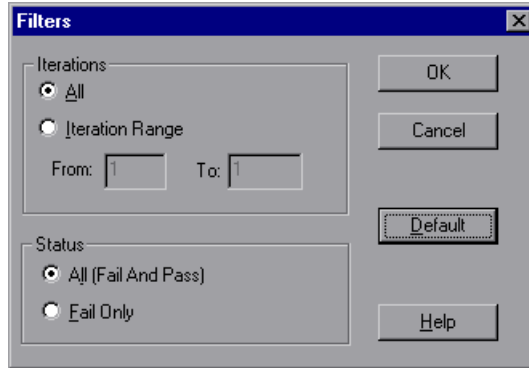
- To collapse a branch, click the Collapse (-) sign to the left of the branch you want to collapse. The report tree hides the details of the branch, and the Collapse sign changes to an Expand (+) sign.
- To collapse all the branches in the report tree, select **View > Collapse All**.
- To expand a branch, click the Expand (+) sign to the left of the branch you want to expand. The report tree displays the details of the branch, and the Expand sign changes to a Collapse (-) sign.
- To expand all the branches in the report tree, select **View > Expand All**.

Filtering Report Information

You can filter the information that is displayed in a VuGen Results Summary report. The filter can be based either on the iteration number or on the status of the iteration.

To filter the information contained in your report:

- 1 Click the **Filter** button on the **Report** toolbar, or select **View > Filters**. The Filters dialog box opens.



- 2 Set the desired filter options. The default filter options are **All**, as shown in the above example.

To limit the report to a specified range of iterations, select **Iteration Range** in the **Iterations** section, and specify a range in the **From** and **To** boxes.

To limit the report to iterations that failed, select **Fail Only** in the **Status** section.

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

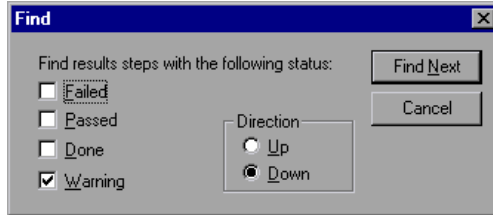
Searching Your Results

You can search for result steps within your Test Results, by their final status: *Failed*, *Passed*, *Done*, or *Warning*. You can select more than one status for your search.

To search for a step with a specific status:



- 1 Select **Tools > Find**, or click the **Find** button on the **Report** toolbar. The Find dialog box opens.



- 2 Select the status (one or more) of the step that you want to find.
- 3 Select a search direction, **Up** or **Down**.
- 4 Click **Find Next**. The cursor jumps to the first match.



- 5 To repeat the search, click the **Find Next** button.

Managing Execution Results

You use the commands in the File menu to open, print, and exit Results Summary reports.

For details on setting Results Summary report options, see “Using VuGen’s Debugging Features for Web Vuser Scripts” on page 144 of Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

Opening a Results Summary Report

When you run a Web Vuser script, VuGen saves the Results Summary report files in a results subfolder of the script folder. The report file has the format: *script_name.qtp*.

To open a Results Summary report:



- 1 Select **File > Open**, or click the **Open** button on the **Report** toolbar. The Open dialog box opens.
- 2 Select the name of the report file that you want to open, and click **Open**.

- 3 To open a recently viewed report, select it from the report history list on the **File** menu.

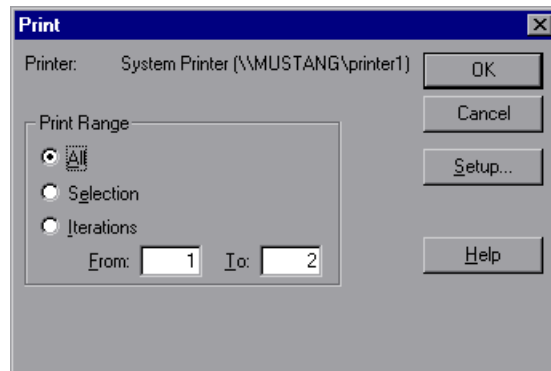
Printing Report Results

You can print a VuGen Results Summary report.

To print a Results Summary report:



- 1 Select **File > Print**, or click the **Print** button on the **Report** toolbar. The Print dialog box opens.



- 2 Select a range from the **Print Range** box:
 - All**—prints the entire report. This includes the Web page for each step in an iteration.
 - Selection**—prints the selected branch in the **Report** tree.
 - Iterations**—prints the specified range of iterations. Specify the range in the **From** and **To** boxes.
- 3 Click **OK** to print.
- 4 To change your printer's setup options, select **File > Print Setup**, and change the settings in the Print Setup dialog box.

Closing a Results Summary Report

To close a Results Summary report, select **File > Exit**. The Test Results window closes.

44

Power User Tips for Web Vusers

This chapter answers some of the questions that are asked most frequently by *advanced users* of Web Vusers. The questions and answers are divided into the following sections:

- Security Issues
- Handling Cookies
- The Run-Time Viewer (Online Browser)
- Browsers
- Configuration Issues

The following information applies to Web Vuser scripts.

Security Issues

Question 1: Do Web Vusers support both secure (HTTPS) and unsecure (HTTP) transactions?

Answer: Yes, Web Vusers do support both secure (HTTPS) and unsecure (HTTP) transactions.

Question 2: Do Web Vusers support digital certificates?

Answer: Yes, Web Vusers support client-side digital certificates. A digital certificate is an attachment to an electronic message used for security purposes. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.

VUGen supports client-side certificates with the following limitations:

- **Recording:** The client certificates are always taken from the IE database, regardless of the actual browser used during the recording. Therefore, if you record using a browser or application other than IE, you must first export the certificate from the recording browser and import it into IE. When importing a certificate into IE, be sure to make its private key exportable:
- **Recording:** In earlier versions of VuGen, `web_set_certificate` was generated whenever a client certificate was used. This function has only one argument: the ordinal number of the certificate in the certificate list. This function can be only be replayed in WinInet mode.

In newer versions of VuGen, `web_set_certificate_ex` is generated. This function has an additional parameter—the path of the file containing the certificate. The certificate file is generated automatically during recording and is saved with the Vuser script. Whenever using WinInet replay mode, the first parameter is used. For socket replay (default), the second parameter is used (certificate file). Note, that if the particular certificate cannot be dumped, for example, if its private key is not exportable, `web_set_certificate_ex` is generated without a file name. In this case, only WinInet replay mode should be used.

Replay: If `web_set_certificate_ex` is used and it has filename argument, it can be used only with socket replay and does not require any custom configuration on the load machines. If `web_set_certificate` is used, or `web_set_certificate_ex` without file name, it can be used only with WinInet based replay. In this case, you need to install all the certificates you have on the recording machine **in the same order** as they appear in its certificate list. This is done through export/import.

Question 3: When I record a Vuser script that accesses an SSL-enabled site, a number of pop-up warning messages appear. Should these messages appear? If so, what do I do with them?

Answer: In order to be able to record access to SSL-enabled sites, VuGen provides its own server certificate instead of the original server certificate. This causes two security violations:

- The certificate that is issued is not for the site to which the user is connecting.

- The certificate is issued by an unknown authority.

These security violations cause the recording browser to display the pop-up warnings messages.

If you are using Netscape 3.0 or higher, or Internet Explorer 4.0 or higher, then you have the option of ignoring these warnings. You can safely ignore the messages.

Note: The pop-up messages appear only when you record the script, not when you execute it.

You can suppress some of the pop-up messages—not all of them.

Question 4: I am using a Web application other than IE and Netscape. When I access a secure site without a recognized certificate, the application automatically aborts. Can I record this application?

Answer: When you access a secure site without a recognized certificate, IE and Netscape issue a warning. Certain browsers and applications do not issue a warning for unrecognized certificates—they simply exit the secure site. To record these sites you must obtain the *pem* file(s) of the certificate and key, and add it to the *certs* directory under the ProTune *bin* folder. List the *pem* files to the *index.txt* file in a format similar to the existing entries: a section name with the hostname and port followed by the name of the *pem* file(s).

```
[demoserver:443]
Certfile=xxx.pem
Keyfile=yyy.pem
```

Question 5: Does VuGen support 128-bit encryption?

Answer: Yes, but you must install a US-only version of Internet Explorer on the recording machine and on any load generators that run the Vuser scripts. Both Netscape and Internet Explorer use 40-bit encryption keys in the international versions of their browsers.

Question 6: Does VuGen support client-side certificates for Internet Explorer?

Answer: Yes, VuGen supports client-side certificates for Internet Explorer.

Question 7: Does VuGen support client-side certificates for Netscape?

Answer: No, VuGen supports client-side certificates only for Internet Explorer. If you have only Netscape certificates, first export the required certificates from Netscape, and then import them into Internet Explorer. Make sure to export and import the certificates in the same order. You must repeat this process on every computer that will record or run a Web Vuser script that requires a certificate.

Question 8: If I look at a Web Vuser script, can I tell whether the Vuser accesses a regular (HTTP) server or an SSL-enabled (HTTPS) server?

Answer: Sometimes. Web Vuser scripts do not distinguish between secure requests and non-secure requests: Graphical Vuser scripts use the same icons for secure requests and non-secure requests; text-based Vuser scripts use the same functions for secure requests and non-secure requests. However, if a step in a Vuser script contains a URL, then you may be able to distinguish from the URL whether the step accesses a regular (HTTP) server or an SSL-enabled (HTTPS) server.

Question 9: What types of authentication do Web Vusers support?

Answer: Web Vusers support Basic authentication and NTLM authentication (NT challenge response authentication).

Handling Cookies

Question 10: Does VuGen handle cookies when I record a Vuser script?

Answer: VuGen automatically handles all cookies that are set via HTTP headers. However, VuGen is unable to always correctly handle cookies that are set by JavaScripts or <meta- > tags. See Question 14 for details.

Question 11: When I run a Web Vuser script, does the Vuser reuse the same cookies that were used when I recorded the Vuser script?

Answer: Yes and No, depending on the type of cookie. Cookies can be divided into two categories: persistent cookies and session cookies:

persistent cookies Text-only strings that identify you to a Web server, and are valid for a limited time period. Persistent cookies are stored on your hard disk.

session cookies Text-only strings that identify you to a Web server only during your current visit (session). Session cookies are not stored on your hard disk.

When you record a Web Vuser script, VuGen detects all cookies that are sent to your browser. VuGen distinguishes between persistent cookies and session cookies as follows:

persistent cookies VuGen records the details of persistent cookies directly into the Vuser script. VuGen uses `web_add_cookie` to include a persistent cookie in a Vuser script. When you run the Vuser script, the Vuser uses these persistent cookies when required.

session cookies VuGen does not save the session cookies that are used during the recording session. Instead, the session cookies are cached while you record, and are then discarded when you stop recording.

When you run the Vuser script, the Vuser uses new session cookies that it receives from the Web server. That is, Vusers do not re-use the same session cookies that were generated when the script was recorded. The session cookies are stored in the Vusers cookie cache, and are then discarded when the Vuser stops. The Vuser does not save these session cookies.

Question 12: Does each Vuser have its own unique cookie cache?

Answer: Yes, each Vuser has its own unique cookie cache—session cookies are not shared, even if the Vusers are running on the same load generator.

Question 13: Must I parameterize the cookies in my recorded Vuser script before I can run the script?

Answer: Sometimes. As described in Question 11, VuGen copies persistent cookies into the Vuser script when you record the script. When you run the Vuser script, the Vuser uses the recorded persistent cookies. If each Vuser requires a unique persistent cookie, then you need to parameterize the cookies in your Vuser script.

Question 14: Do Web Vusers handle cookies that are set inside JavaScripts?

Answer: VuGen automatically handles all cookies that are set via HTTP headers. However, VuGen is unable to always correctly handle cookies that are set by a JavaScript contained in an HTML page. Cookies that are set via JavaScripts create unique problems during recording and replay:

Recording

VuGen should record persistent cookies—not session cookies—into a Vuser script (via **web_add_cookie** statements). However, due to technological constraints, all cookies that are set by JavaScripts are recorded by VuGen as persistent cookies—even if the cookies are session cookies.

Workaround: After recording a Vuser script, insert correlation statements to correlate all **web_add_cookie** statements that set session cookies. Do not delete **web_add_cookie** statements that set persistent cookies.

Replay

Web Vusers do not run JavaScripts that are embedded inside HTML pages. Therefore any session cookies that are created by such JavaScripts are not created when the Vuser runs.

Workaround: After recording a Vuser script, insert correlation statements into the script to determine the appropriate cookies. Then insert **web_add_cookie** statements into the Vuser script to set the appropriate cookies.

Question 15: Can a Vuser manipulate cookies during run-time?

Answer: Yes, while a Vuser is running, the Vuser can manipulate the cookies that are stored in its cookie cache. You can use the following functions in a Vuser script to manipulate the cookie cache:

- `web_add_cookie()`
- `web_remove_cookie()`
- `web_cleanup_cookies()`

Refer to the *Online Function Reference* (**Help > Function Reference**) for details on the above functions.

The Run-Time Viewer (Online Browser)

Question 16: How does the run-time viewer display Web pages?

Answer: When you run a Web Vuser script, the Web servers accessed by the Vuser download information to the Vuser. This information is usually in HTML format. The Vuser saves this information to the Vuser's results directory. Each Web page is saved in HTML format as a separate *.htm* file. While the Vuser runs, the run-time viewer loads the *.htm* files that are saved in the Vuser results directory, and displays the resulting Web pages.

Question 17: JavaScript errors frequently appear when I use the run-time viewer. What causes this, and what can I do to prevent it?

Answer: When you use the run-time viewer, make sure that the **Enable Scripting** option from the Runtime Browser's **Options** menu is not checked. This instructs the run-time viewer not to run any JavaScripts and ensures that JavaScript errors no longer appear in your run-time viewer.

As described in the answer to Question 16, when you run a Vuser script, VuGen saves the information that is returned by the server. The run-time viewer displays this saved information—not the information that is returned directly by the server.

Question 18: What types of data can the run-time viewer display?

Answer: The run-time viewer can display HTML pages only. It cannot display any other information types.

Question 19: When I run a Vuser script, why does the run-time viewer not display the data that the Vuser submits to the Web server?

Answer: The run-time viewer shows only the HTML page that is returned by the server to the Vuser. The run-time viewer does not show any data that the Vuser submits to the Web server. For further details, see the answer to Question 16.

Question 20: Does the run-time viewer correctly display multi-window applications?

Answer: No, the run-time viewer currently does not correctly display multi-window applications.

Browsers

Question 21: Why is it recommended that I have Internet Explorer 4.0 or higher installed on my computer—even if I always use Netscape to record my scripts?

Answer: VuGen relies heavily on WinInet, the Microsoft Internet API. This applies to both recording and replaying Web Vuser scripts. The WinInet.dll is the Microsoft infrastructure for Internet connections.

ProTune installs version 3.0 of the WinInet.dll—unless a newer version is already installed on the computer. Version 3.0 has many limitations. Version 4.0 is far superior, so we recommend that you install version 4.0 for best results with Web Vusers. The simplest legal way to install WinInet.dll version 4.0 is to install Internet Explorer 4.0 or higher.

Question 22: If I install Internet Explorer 3.0 and not Internet Explorer 4.0 or higher, what features will I not be able to use?

Answer: Internet Explorer includes the WinInet.dll. You require the version 4 of the WinInet.dll file to enable the following features:

- Proxy authentication
- NTLM authentication (NT challenge response authentication)
- Digital certificates
- Run-Time Browser
- Reports
- WAP recording

Question 23: Must I use a standard browser—such as Netscape or Internet Explorer—when I record?

Answer: No, you can use any browser of your choice when you record a Web Vuser script. In fact, you do not need to use a browser. Instead you can use any application that generates HTTP(S) requests. The only requirement of the application is that you must be able to set the proxy settings to localhost:7777 so that VuGen can record the HTTP(S) requests.

Question 24: How do I record a non-standard HTTP(S) application?

Answer: Perform the following procedure:

- 1** Choose **Tools > Recording Options >** and click the **Browser** tab. Select **Manually launch a browser**.
- 2** Click the **Recording Proxy** tab and clear the **Obtain the proxy settings from the recording browser** check box. Specify the proxy settings (if applicable).
- 3** Click the **Start Recording** button. VuGen prompts you for the proxy settings required for the recorded application.
- 4** Perform the required changes in the application being recorded.
- 5** Begin recording the session.
- 6** Close the application when you are finished recording and restore the original proxy settings (failure to do so may prevent it from working).

Question 25: Does VuGen ever modify any of the proxy settings in my recording browser?

Answer: Yes. When you start to record a Web Vuser script, VuGen launches the browser that you specified. VuGen then directs the browser to go through the VuGen proxy server. To do this, VuGen modifies the proxy settings on the recording browser. VuGen changes the proxy setting to localhost:7777 immediately, by default. After recording, VuGen restores the original proxy settings to the recording browser. You must not change the proxy settings while VuGen is recording.

Question 26: My browser crashed while I was recording. I can now not access any sites with my browser—even if I do not record. Why not?

Answer: The answer to Question 25 describes how VuGen changes the proxy settings in your browser during recording. If your browser crashes while you record, VuGen may not be able to restore your original proxy settings for your browser. Your browser will then still have the localhost:7777 setting—which prevents it from accessing any sites. You must manually restore the original proxy settings for your browser.

Question 27: Does VuGen support Socks proxies?

Answer: Yes, VuGen does support Socks proxies. To use a Socks proxy you must use Internet Explorer—not Netscape—as the recording browser. In addition:

- Use Internet Explorer 4.0 or higher to define the Socks proxy.

In Internet Explorer, select **View > Internet Options**. Click the **Connection** tab, and then click **Advanced** in the **Proxy Server** group. In the Proxy Settings dialog box, enter the appropriate Socks proxy server settings.

This step applies to the computer that you use to record the Vuser scripts, as well as to all the computers that will run Vusers that access the Socks proxy server.

- Define Internet Explorer as the default browser.

You can do this by associating all files that have an *.htm* extension with Internet Explorer.

This step applies to the computer that you use to record the Vuser scripts, as well as to all the computers that will run Vusers that access the Socks proxy server.

- Instruct VuGen to take the proxy settings from the recording browser when you record a Vuser scripts.

In VuGen, select **Tools > Recording Options**. Click the **Recording Proxy** tab. Select the **Obtain the proxy setting from the recording browser** check box.

This step applies only to the computer that you use to record the Vuser scripts—not to the computers that will run the Vusers.

- Instruct all Vusers that run the script to obtain the proxy setting from the default browser.

In VuGen, select **Vuser > Run Time Settings**. Click the **Proxy** tab, and select the **Obtain the proxy setting from the default browser** option.

This setting applies to all Vusers that run the Vuser script.

Question 28: If I have Netscape installed—and not Internet Explorer—can I display execution reports?

Answer: In order for VuGen to display execution reports, you need Internet Explorer, Version 4.0 or higher.

Question 29: I noticed that the **Number of Concurrent Connections** Run-Time setting is no longer available. Can I still modify this setting?

Answer: Yes. You modify this setting using the `web_set_sockets_options` function. To set the maximum number of connections per host, use the `MAX_CONNECTIONS_PER_HOST` flag and assign it the desired value. To set a global number of connections, the maximum number of simultaneous connections per Vuser, use the `MAX_TOTAL_CONNECTIONS` flag and set it to the desired number. The default number of concurrent connections when using Internet Explorer is four for HTTP 1.0 and two for HTTP 1.1. For more information, see the `web_set_sockets_options` in the function reference.

Configuration Issues

Question 30: I performed a snapshot comparison and the results were very inaccurate.

Answer: Check the Correlation tab in the General Options dialog box (**Options > General**). Make sure the Comparison mode is set to HTML—not Text. The Text comparison mode is only applicable to non-HTML snapshots.

Part IX

Enterprise Java Bean Protocols

45

Performing EJB

The Enterprise Java Beans (EJB) testing tool generates scripts for tuning EJB objects.

This chapter describes:

- ▶ Working with the EJB Detector
- ▶ Understanding EJB Vuser Scripts
- ▶ Creating an EJB Testing Vuser
- ▶ Setting EJB Recording Options
- ▶ Running EJB Vuser Scripts

Refer to Appendix B, “EJB Architecture and Testing” for additional information about EJB testing.

The following information only applies to EJB Testing Vuser scripts.

About EJB Testing

VuGen provides several tools for developing a script that tests Java applications. For generating a Vuser script through recording, use the Jacada, CORBA or RMI Vusers. For creating a script through programming, use the custom Java Vusers.

EJB Testing Vusers differ from the standard Java Vusers in that VuGen automatically creates a script to tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. ProTune’s EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to tune, and ProTune generates a script that

tests 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 591.

Working with the EJB Detector

The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

Installing the EJB Detector

The EJB Detector can be installed and invoked on the application server's machine or alternatively, on the client machine. To run the EJB Detector on the client machine you must have a mounted drive to the application server machine.

To install the EJB detector agent:

- 1** Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
- 2** Unzip the `<LR_root>\ejbcomponent\ejbdetector.jar` file into the EJB Detector directory.

Running the EJB Detector

The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the

application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

The EJB Detector can run from the command-line, or from a batch file.

To run the EJB Detector from the command line:

- 1** Before running the EJB Detector from the command line, add the `DETECTOR_HOME\classes` and the `DETECTOR_HOME\classes\xerces.jar` to the `CLASSPATH` environment variable.
- 2** If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the `CLASSPATH`:

For WebLogic 4.x: `<WebLogic directory>\lib\weblogicaux.jar`

For WebSphere 3.x: `<WebSphere directory>\lib\ujc.jar`

- 3** If your EJBs use additional classes directory or .jar files, add them to the `CLASSPATH`.
- 4** To run the EJB Detector from the command-line, use the following string:
`java EJBDetector [search root dir] [listen port]`

search root dir

one or more directories or files in which to search for EJBs (separated by semicolons). Follow these guidelines:

BEA WebLogic Servers 4.x and 5.x: Specify the application server root directory.

BEA WebLogic Servers 6.x: Specify full path of the domain folder.

WebSphere Servers 3.x: Specify the full path of the deployed EJBs folder.

WebSphere Servers 4.0: Specify the application server root directory.

Oracle OC4J: Specify the application server root directory.

Sun J2EE Server: Specify the full path to the deployable *.ear* file or directory containing a number of *.ear* files.

If unspecified, the classpath will be searched.

listen port

The listening port of the EJB Detector. The default port is 2001. If you change this port number, you must also specify it in the **Host name** box of the **Generate EJB Test** dialog box.

For example, if your host is *metal*, if you are using the default port, you can specify *metal*. If you are using a different port, for example, port 2002, enter *metal:2002*.

To run the EJB Detector from a batch file:

You can launch the EJB detector using a batch file, *EJB_Detector.cmd*. This file resides in the root directory of the EJB Detector installation, after you unzip *ejbdetector.jar*.

- 1 Open *env.cmd* in the EJB Detector root directory, and modify the following variables according to your environment:

<i>JAVA_HOME</i>	the root directory of JDK installation
<i>DETECTOR_INS_DIR</i>	the root directory of the Detector installation
<i>APP_SERVER_DRIVE</i>	the drive hosting the application server installation
<i>APP_SERVER_ROOT</i>	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 <i>.ear</i> file or directory containing a number of <i>.ear</i> 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.

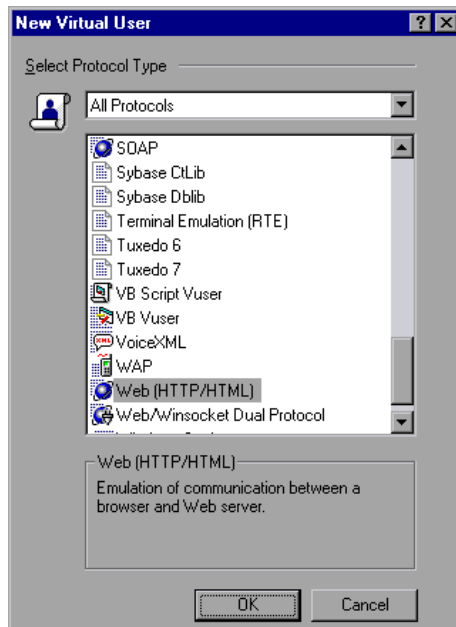
After the EJBs are detected, the HTTP Server is initialized and waits for requests from the VuGen EJB-Testing Vuser. If there are problems in this communication process, enable the property `webserver.enableLog` in the `webserver.properties` file located in the `DETECTOR_HOME\classes` directory.

This enables printouts of additional debug information, and other potentially important error messages in the `webserver.log` file.

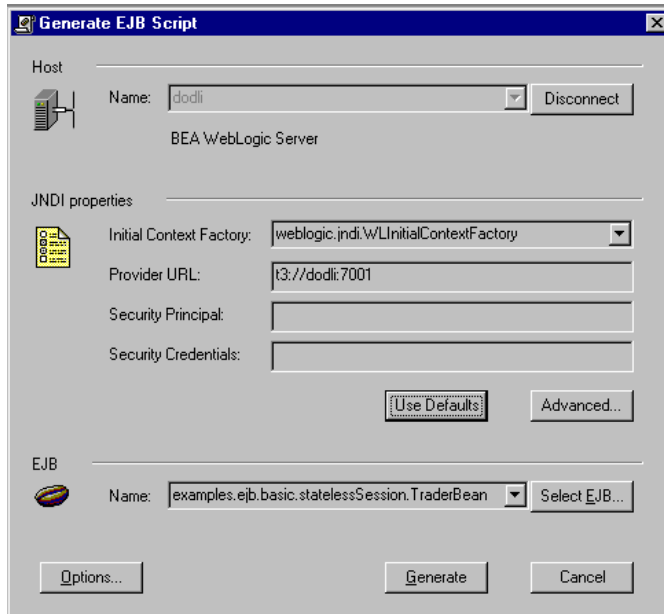
Creating an EJB Testing Vuser

To create an EJB Vuser script:

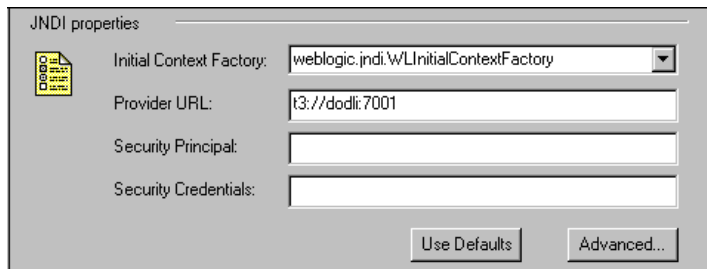
- 1 Choose **File > New** or click the **New** button. The New Virtual User dialog box opens.



- 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 the EJB Detector is installed. Note that the Detector must be running in order to connect. Click **Connect**. The **JNDI properties** section is enabled.



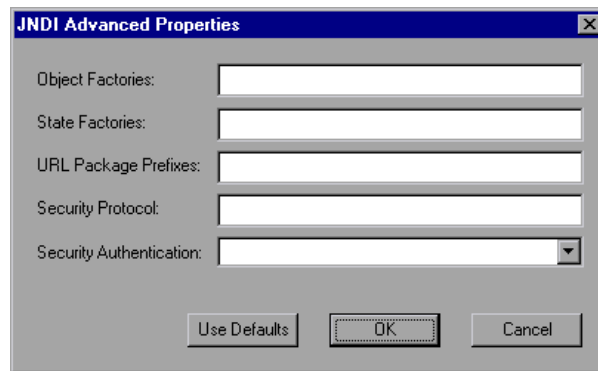
- 4 The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory** and the **Provider URL**.

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:

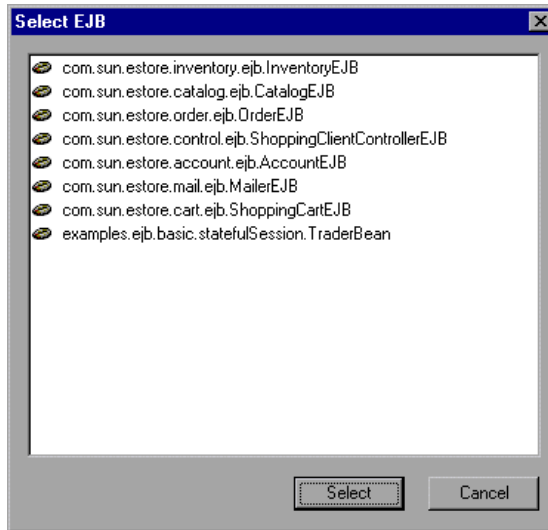
Type	Initial Context Factory	Provider URL
WebLogic	weblogic.jndi.WLInitialContextFactory	t3://<appserver_host>:7001
WebSphere 3.x	com.ibm.ejs.ns.jndi.CNInitialContextFactory	iiop://<appserver_host>:900
WebSphere 4.x	com.ibm.websphere.naming.WsnInitialContextFactory	iiop://<appserver_host>:900
Sun J2EE	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 choose 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 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 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.

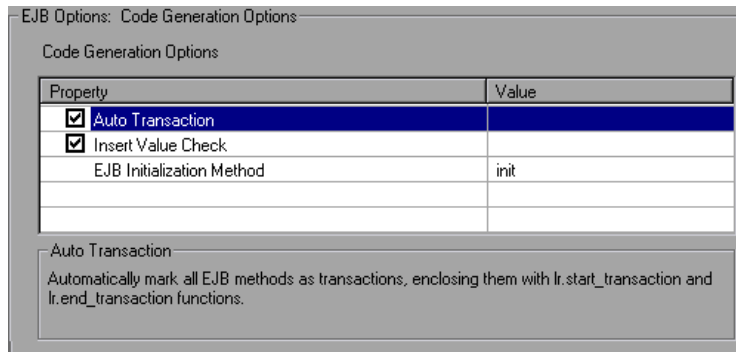
Setting EJB Recording Options

The recording options that are available for EJB Vusers are in the areas of Classpath and Code Generation. For information on the Classpath options, see Chapter 13, "Setting Java Recording Options."

The **EJB Code Generation** options allow you to set properties in the area of automatic transactions and value checks. You can also indicate where to store the initialization method.

To set the **EJB Code Generation** recording options:

- 1 Click **Options** in the Start Recording dialog box. Select the **EJB Options:Code Generation Options** node in the Recording Options tree to edit the code generation options.



- 2 Enable the **Auto Transaction** option to automatically mark all EJB methods as transactions. This encloses all methods with `lr.start_transaction` and `lr.end_transaction` functions. By default, this option is enabled (true).
- 3 Enable the **Insert Value Check** option to automatically insert an `lr.value_check` function after each EJB method. This function checks for the expected return value for primitive values and strings.
- 4 Choose an **EJB Initialization Method**. This is the method to which the EJB/JNDI initialization properties are written. The available methods are *init* (default) and *action*.

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://beallogic: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 may choose to 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, ensure 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 589).
- 3** Insert additional methods, and modify their default values. (refer to the section on inserting Java functions in Chapter 12, “Recording Java Language Vuser Scripts.”).
- 4** Set the general run-time settings for the script. For more information, see Chapter 9, “Configuring Run-Time Settings.”
- 5** Set the Java VM run-time settings: Specify all additional classpaths and additional VM parameters. Make sure to include your application server EJB classes. The actual classes of the EJB 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 15, “Configuring Java Run-Time Settings.”
- 6** For **Websphere 3.x** users:

Using the IBM JDK 1.2 or higher:

- Add the `<WS>\lib\ujc.jar` to the classpath.

Using the Sun JDK 1.2.x:

- ▶ Remove the file <JDK>\jre\lib\ext\iimp.jar
- ▶ Copy the following files from the <WS>\jdk\jre\lib\ext folder to the <JDK>\jre\lib\ext directory: iioprt.jar, rmiorb.jar.
- ▶ Copy the 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 choose **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 verify that your EJB is functional, you can tune it by assigning it to multiple Vusers in session steps. For more information, see the *ProTune Console User's Guide*.

Part X

ERP/CRM Protocols

46

Creating Oracle NCA Vuser Scripts

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 describes:

- ▶ Getting Started with Oracle NCA Vusers
- ▶ Recording Guidelines
- ▶ Enabling the Recording of Objects by Name
- ▶ Using Oracle NCA Vuser Functions
- ▶ Understanding Oracle NCA Vusers
- ▶ Switching Between Tree View and Script View
- ▶ Configuring the Run-Time Settings
- ▶ Testing Oracle NCA Applications
- ▶ Correlating Oracle NCA Statements for Load Balancing
- ▶ Recording in Pragma Mode

The following information applies only to the Oracle NCA protocol.

About Creating Oracle NCA Vuser Scripts

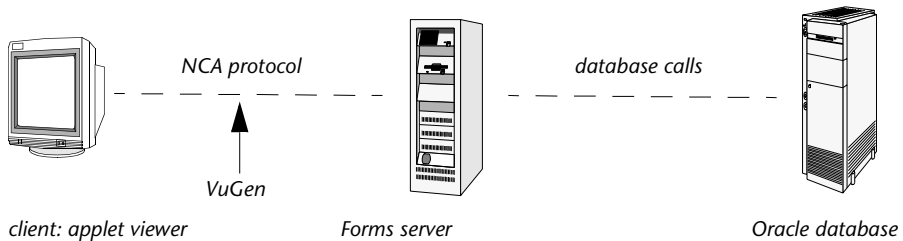
Oracle NCA is a Java-based database protocol. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer. This eliminates the need for client software and allows you to perform database actions from all platforms that

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 (Oracle 8.x).

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 Vuser dialog box. For more information, see Chapter 3, "Recording with VuGen."

Getting Started with Oracle NCA Vusers

The following procedure outlines how to create an Oracle NCA Vuser script.

1 Ensure 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 your version of Oracle Forms is supported by ProTune. For more information, see “Recording Guidelines,” on page 600 and the Read me file.

2 Create a skeleton Oracle NCA Vuser script.

Use VuGen to create a skeleton Oracle NCA Vuser script. For details, see “Vuser Script Sections,” on page 26.

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 Vuser Scripts,” on page 28.

4 Enhance the Vuser script.

Use the Insert menu to add transactions, rendezvous points, comments, and messages in order to enhance the Vuser script. For details, see Chapter 6, “Enhancing Vuser Scripts.”

5 Parameterize the script.

Replace recorded constants with parameters. For details, see Chapter 7, “Defining Parameters.”

6 Set the run-time properties for the script.

Configure run-time settings for the Vuser script. The run-time settings define certain aspects of the script execution. For details, see Chapter 9, “Configuring Run-Time Settings.”

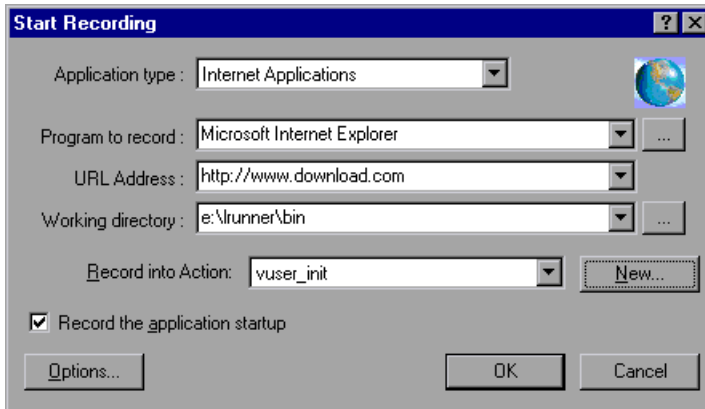
7 Save and run the Vuser script.

Run the script from VuGen and view the execution log for run-time information. For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

Recording Guidelines

When recording an Oracle NCA Vuser script, follow these guidelines:

- ▶ Specify which browser VuGen should use when recording an Oracle NCA session. In the Start Recording dialog box, select the desired browser in the **Program to Record** list. The list contains all of the available browsers.



- ▶ Close all browsers before you begin recording.
- ▶ Record the login procedure in the *vuser_init* section. Record a typical business process in the *Actions* section. When you run the script, you can then specify multiple iterations for a specific business process. For more information, see “Recording Vuser Scripts,” on page 28.

```
vuser_init()
{
connect_server("199.203.78.170", "9000"/*version=107*/,"
    module=e:\appsnc\ \fnd\ \7.5\ \forms\ \us\ \fndscsgn
    userid=applsypub/pub@vision fndnam=apps");
edit_set("FNDSCSGN.SIGNON.USERNAME.0","VISION");
edit_set("FNDSCSGN.SIGNON.PASSWORD.0","WELCOME");
button_press("FNDSCSGN.SIGNON.CONNECT_BUTTON.0");
lov_retrieve_items("Responsibilities",1,17);

    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. In Oracle Forms, the application server uses the *Forms Listener Servlet* to create a runtime process for each client. The runtime process, known as the *Forms Server Runtime* process, 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 6 or 9 support, restore the original settings.

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/visappl/fnd/11.5.0/forms/US/FNDSCSGN userID=APPLSYSPUB/PUB@VIS
fndnam=apps record=names ");
```

If the **record=names** argument does not appear in the **nca_connect_server** function, you 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
- Configuration file

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">
```

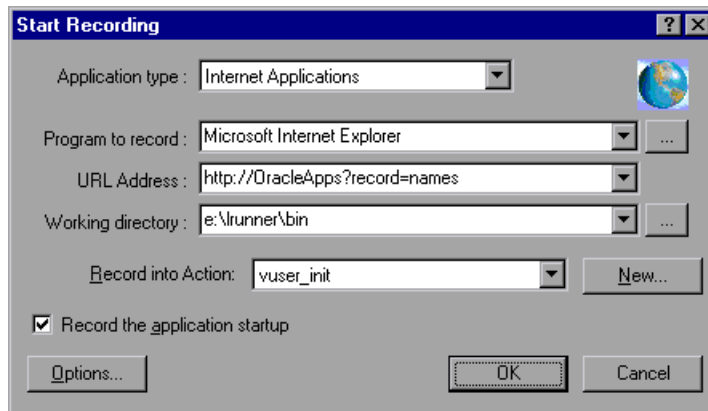
and add the Oracle key "record=names":

```
<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=%use-  
rid% %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 *basejini.htm* file add the following line to the parameter definitions;

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the <EMBED> tag, add the following line:

```
...  
serverApp="%serverApp%"  
logo="%logo%"  
imageBase="%imageBase%"  
formsMessageListener="%formsMessageListener%"  
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file *formsweb.cfg*, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the *basejini.htm* file and store it at another location. In the servlet configuration file, edit the *baseHTMLJinitiator* parameter to point to the new file.

Using Oracle NCA Vuser Functions

VuGen automatically records most of the functions listed in this section while you perform typical NCA business processes. The functions are recorded with an **nca** prefix. (NCA functions recorded without **nca** prefixes in earlier versions of VuGen, are still supported.) You can also manually program any of the functions into your Vuser script. When working in tree view, click the graphical icon for the appropriate step. In text view, you can manually add the desired function. For more information about the Oracle NCA Vuser functions, refer to the *Online Function Reference* (**Help > Function Reference**).

Button Object Functions

<code>nca_button_double_press</code>	Performs a double press on a push button.
<code>nca_button_press</code>	Activates a push button.
<code>nca_button_set</code>	Sets the state of the specified button.

Combo Box Object Functions

<code>nca_combo_select_item</code>	Selects an item in a combo box.
<code>nca_combo_set_item</code>	Sets a new item in a combo box.

Connection Functions

<code>nca_connect_server</code>	Connects to an Oracle NCA server.
<code>nca_logon_connect</code>	Performs a login to an Oracle NCA database.
<code>nca_logon_cancel</code>	Disconnects from an Oracle NCA database.

Edit Object Functions

<code>nca_edit_box_press</code>	Clicks on an edit box message.
<code>nca_edit_click</code>	Clicks in an edit object.
<code>nca_edit_get_text</code>	Returns the text in an edit object.
<code>nca_edit_press</code>	Activates the browse button in an edit field.
<code>nca_edit_set</code>	Replaces the entire contents of an edit object.

Flex Object Functions

<code>nca_flex_click_cell</code>	Clicks a table cell in a Flexfield window.
<code>nca_flex_get_cell_data</code>	Gets data from a Flexfield cell.
<code>nca_flex_get_column_name</code>	Gets the name of a column in a Flexfield window
<code>nca_flex_press_clear</code>	Clicks Clear in a Flexfield window.
<code>nca_flex_press_find</code>	Clicks Find in a Flexfield window.
<code>nca_flex_press_help</code>	Clicks Help in a Flexfield window.
<code>nca_flex_press_lov</code>	Clicks on the List of Values button in a Flexfield window.
<code>nca_flex_press_ok</code>	Clicks OK in a Flexfield window.
<code>nca_flex_set_cell_data</code>	Inserts data in a Flexfield window.
<code>nca_flex_set_cell_data_press_ok</code>	Clicks OK in a Flexfield window after data is entered.

List Item Functions

<code>nca_list_activate_item</code>	Activates an item in a list (double-click).
<code>nca_list_select_index_item</code>	Selects a list item by its index.
<code>nca_list_select_item</code>	Selects a list item by its name.
<code>nca_lov_auto_select</code>	Specifies the first letter of an item.
<code>nca_lov_find_value</code>	Clicks Find in a List of Values window.
<code>nca_lov_get_item_name</code>	Retrieves the name of an entry in a list of values by the entry's index number.
<code>nca_lov_retrieve_items</code>	Retrieves a list of values.
<code>nca_lov_select_index_item</code>	Selects an item from a list of values by its index number.
<code>nca_lov_select_item</code>	Selects an item from a list of values.
<code>nca_lov_select_random_item</code>	Selects a random item from a list of values.

Java Object Functions

<code>nca_java_action</code>	Performs an event on a Java object.
<code>nca_java_get_value</code>	Retrieves the value of a Java object.
<code>nca_java_set_reply_property</code>	Sets a reply property for a Java object.

Menu Object Functions

<code>nca_menu_select_item</code>	Selects an item from a menu.
-----------------------------------	------------------------------

Message Functions

<code>nca_popup_message_press</code>	Clicks a button in a popup window.
<code>nca_message_box_press</code>	Clicks a button in a message window.

Object Functions

<code>nca_obj_get_info</code>	Returns the value of an object property.
<code>nca_obj_mouse_click</code>	Clicks within an object.
<code>nca_obj_mouse_dbl_click</code>	Double-clicks within an object.
<code>nca_obj_status</code>	Returns the status of the specified object.
<code>nca_obj_type</code>	Types special characters into an edit box.

Response Object Functions

<code>nca_response_press_lov</code>	Clicks a drop down arrow in a Response box.
<code>nca_response_press_ok</code>	Clicks OK inside a Response box.
<code>nca_response_set_cell_data</code>	Inserts data into a cell in a Response box.
<code>nca_response_set_data</code>	Inserts data into a Response box.

Scroll Object Functions

<code>nca_scroll_drag_from_min</code>	Drags the scroll to the specified distance from the minimum position (0).
<code>nca_scroll_line</code>	Scrolls the specified number of lines.
<code>nca_scroll_page</code>	Scrolls the specified number of pages.

Session Functions

<code>nca_console_get_text</code>	Retrieves the console message
<code>nca_set_iteration_offset</code>	Sets an offset value for an object ID.
<code>nca_set_server_response_time</code>	Sets the server response time.
<code>nca_set_exception</code>	Specifies how to handle exceptions.
<code>nca_set_think_time</code>	Sets the think time range.

Tree Object Functions

<code>nca_tree_activate_item</code>	Activates an item in an NCA tree.
<code>nca_tree_collapse_item</code>	Collapses a tree item.
<code>nca_tree_expand_item</code>	Expands a tree item.
<code>nca_tree_select_item</code>	Selects an item in an NCA tree.

Window Object Functions

<code>nca_win_get_info</code>	Returns the value of an window property.
<code>nca_win_close</code>	Closes a window.
<code>nca_set_window</code>	Indicates the name of the current window.

You can further enhance your script with C Vuser functions such as **lr_output_message** and **lr_rendezvous**. For information on using these functions, see Chapter 6, “Enhancing Vuser Scripts.”

Understanding Oracle NCA Vusers

When you create an Oracle NCA Vuser script, VuGen records all of the NCA communication between the client and the application server. While you record, VuGen generates *context sensitive* functions. These functions describe your actions on the database in terms of GUI objects (such as windows, lists, and buttons). As you record, VuGen inserts the context sensitive functions into the Vuser script.

After you finish recording, you can modify the functions in your script, or add additional functions to enhance it. For information about enhancing Vuser script, see Chapter 6, “Enhancing Vuser Scripts.” For a list of the available Oracle NCA Vuser functions, see “Using Oracle NCA Vuser Functions,” on page 605. 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 Opera-
tions");
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://ussciffirms05.sfb.na/servlet/f60servlet\?config
    =mult", LAST);
```

In the above example, the right boundary is \r. The actual right boundary may differ between systems.

Switching Between Tree View and Script View

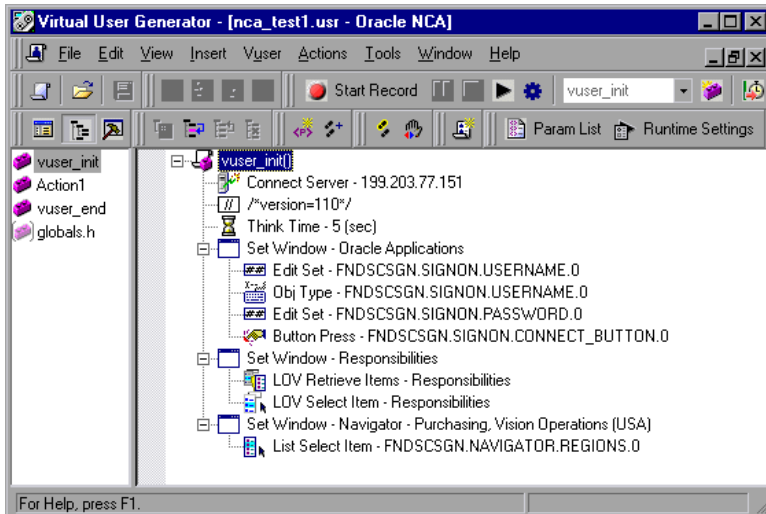
When viewing and editing Oracle NCA Vuser script in VuGen, you choose between viewing the script in the icon-based tree view or the text-based script view.

To display the tree view of an Oracle NCA Vuser script:



From the VuGen main menu, select **View > Tree View**, or click the **View script as tree** icon. The Vuser script is displayed in the icon-based tree view. If you are already in the tree view, the menu item is disabled.

Tree view



To display the script view:

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

Script view

```
#include <orafuncs.h>
vuser_init()
{

    connect_server("199.203.77.151", "9000"/*version=110*/,"module=f:
    lr_think_time(5);

    set_window("Oracle Applications");
    edit_set("FNDSCSGN.SIGNON.USERNAME.0","OPERATIONS");
    obj_type("FNDSCSGN.SIGNON.USERNAME.0","\t",0);
    edit_set("FNDSCSGN.SIGNON.PASSWORD.0",lr_decrypt("393b6d2e00c5857
    button_press("FNDSCSGN.SIGNON.CONNECT_BUTTON.0");

    set_window("Responsibilities");
    lov_retrieve_items("Responsibilities",1,17);
    lov_select_item("Responsibilities","Purchasing, Vision Operations");

    set_window("Navigator - Purchasing, Vision Operations (USA)");
    list_select_item("FNDSCSGN.NAVIGATOR.REGIONS.0","Functions");
    return 0;
}
```

Configuring the Run-Time Settings

Before running your script, you can set the run-time settings to allow the script to accurately emulate a real user. For information on the general run-time settings for all protocols, such as think time, pacing, and logging, see Chapter 9, “Configuring Run-Time Settings.”

The following section describes the run-time settings specific to Oracle NCA Vusers. These run-time settings allow you to indicate the modem speed to emulate and other communication parameters.

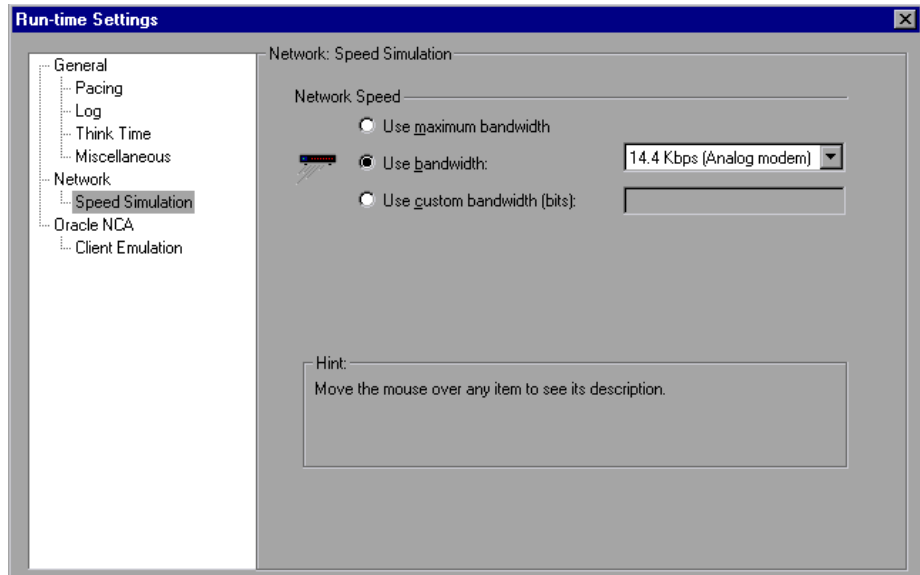
Speed Simulation

For speed emulation, you can select any of the available speed options ranging from 14.4 to 512 kbs or you can specify a custom value. By default, VuGen tries to emulate the maximum available bandwidth.

To set the Speed Simulation settings:



- 1 Open the Run-Time Settings dialog box. Choose **Vuser > Run-Time Settings** or click the **Run-Time Settings** button on the VuGen toolbar. To open the Run-Time Settings dialog box from the ProTune Console, click the **Runtime Settings** button.
- 2 Select the **Network:Speed Simulation** node.



- 3 Set the modem emulation speed:
 - Use the maximum available bandwidth. (default)
 - To specify a standard modem speed emulation, select **Use bandwidth** and choose a bandwidth from the standard list.
 - To specify a custom bandwidth, select **Use custom bandwidth**, and specify the desired bandwidth in bits.

- 4 Click **OK** to accept the settings and run the script.

Client Emulation


You can configure several network settings to accurately emulate an Oracle NCA client. You can set:

Network timeout: The time that an Oracle NCA Vuser waits for a response from the server. The default value of -1 disables the timeout and the client waits indefinitely.

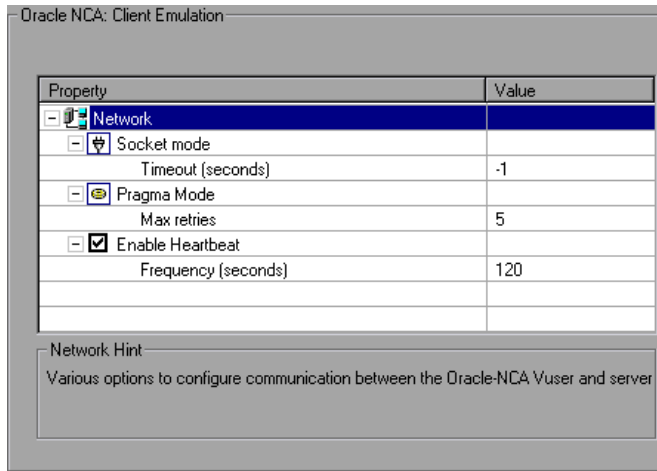
Pragma Mode: In Pragma mode, communication is carried out in the Oracle-defined Pragma mode. This communication level, above the HTTP and Servlet levels, is characterized by the periodic sending of messages. In this mode, the client recognizes that the server cannot respond with data immediately. The server sends messages at given intervals until it is able to send the requested data. In the **Max Retries** section, you indicate the maximum number of *IfError* messages the client will accept from the server before issuing an error. *IfError* messages are the periodic messages the server sends to the client, indicating that it will respond with the data as soon as it is able. For information about recording in this mode, see “Recording in Pragma Mode,” on page 621

Enable Heartbeat: You can enable or disable the heartbeat sent to the Oracle server. The heartbeat verifies that there is proper communication with the sever. If you are experiencing a heavy load on the Oracle NCA server, disable the heartbeat. If you enable the heartbeat, you can set the frequency of how often heartbeat messages are sent to the server.

To set the Client Emulation settings:

- 1  Open the Run-Time Settings dialog box. Choose **Vuser > Run-Time Settings** or click the **Run-Time Settings** button on the VuGen toolbar. To open the Run-Time Settings dialog box from the ProTune Console, click the **Runtime Settings** button.

- 2 Select the **Oracle NCA:Client Emulation** node from the Run-Time settings tree.



- 3 Set the network timeout value in seconds. To instruct the client to wait indefinitely for a server response, use the default value of -1.
- 4 When working in Pragma mode, specify the number of retries **Max Retries**, (*IfError* messages) for the client to accept before issuing an error. The default is 20.
- 5 To enable the sending a a heartbeat to the Oracle NCA server, select the **Enable Heartbeat** option. In the next line, specify a frequency in seconds for the sending of the heartbeat. The default is 120 seconds.
- 6 Click **OK** to accept the settings and run the script.

Testing Oracle NCA Applications

The following sections contain several tips for testing secure Oracle NCA applications and servlets.

Testing Secure Oracle NCA Applications

- When selecting the protocols to record, you only need to select **Oracle NCA**—not **Web Protocol** from the protocol list. VuGen records the security

information internally and therefore does not need the explicit Web functions.

- In the Port Mapping recording options, delete any existing entries for port 443 and create a new entry for the Oracle server name:

Service ID: HTTP

Target Server: Oracle Forms Server IP address or long host name

Target Port: 443

Connection Type: SSL

SSL Version: Active SSL version. If in doubt, select SSL 2/3.

For more information, see Chapter 5, “Configuring the Port Mappings.”

- If you encounter problems when replaying an NCA HTTPS script during the **nca_connect_server** command, insert the following function at the beginning of the script.

```
web_set_sockets_option("SSL_VERSION","3");
```

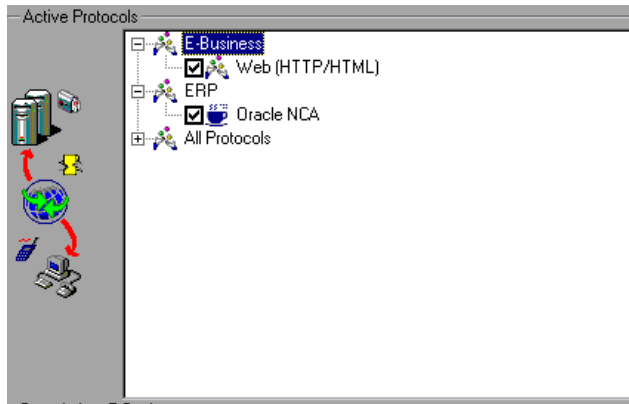
Testing Servlets and other Oracle NCA Applications

Certain NCA sessions use servlets.

- the Forms Listener servlet
- applications or modules that use both NCA and HTTP communications, such as the Oracle Configurator
- the initializing of the NCA application (downloading the applet, jar, and gif files)

When recording servlets, you must record both Oracle NCA and Web functions. You can do this by initially creating a multi-protocol script. Alternatively, if you created a single protocol script for Oracle NCA, open

the **General:Protocols** node in the Recording Options, and enable the Web protocol. Then you can begin recording



If you are unsure whether your application uses servlets, check the *default.cfg* file in the script directory. Locate the entry

UseServletMode=

If the value is 1 or 2, then servlets are being used and you must enable HTTP recording in addition to Oracle NCA.

If you already recorded a script, you can regenerate the code automatically to include the Web functions without having to re-record. Choose **Tools > Regenerate Vuser**, 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 Chapter 5, “Configuring the Port Mappings.”

Correlating Oracle NCA Statements for Load Balancing

ProTune supports load balancing for multiple application servers. You correlate the HTTP return values with the **nca_connect_server** parameters. ProTune 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 Call the web_url function to assign values to serverHost and serverArgs:

```
web_url("step_name", "URL=http://server1.merc-int.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.merc-int.com/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
&RESP_APP=AR&RESP_KEY=RECEIVABLES_MANAGER&SECGRP_KEY=STANDAR
D", LAST);
```

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

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

...

```
web_url("fnd_icx_launch.runforms",
"URL=http://ABC-
123:8002/pls/VIS/fnd_icx_launch.runforms\?ICX_TICKET=<icx_ticket>&RESP_A
PP=AR&RESP_KEY=RECEIVABLES_MANAGER&SECGRP_KEY=STANDARD", LAST);
```

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

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=123.45.789.12" (RelFrameld=1)
vuser_init.c(8):
/servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdroot=my1sanw2
n1.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=<NCA\ServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see “Client Emulation,” on page 613.

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"
  ...

```

47

Creating Baan Vuser Scripts

You use VuGen to develop Baan Vuser script. You record typical Baan sessions with VuGen and you enhance the scripts with Baan Vuser functions.

This chapter describes:

- ▶ Getting Started with Baan Vuser Scripts
- ▶ Baan Vuser Functions
- ▶ Creating a Baan Vuser Script
- ▶ Understanding Baan Vuser Scripts
- ▶ Customizing Baan Vuser Scripts

The following information applies only to Baan Vuser scripts.

About Developing Baan Vuser Scripts

The Baan type Vuser script lets you test your Baan application and test your system under load. VuGen records your entire Baan session, including the login information to the Baan server.

When you record actions, VuGen creates a script using *Context Sensitive* functions. Context Sensitive functions depict actions in the application under test in terms of GUI objects (such as windows, lists, and buttons). Each time you record an operation, a function is generated which describes the object selected and the action performed.

Getting Started with Baan Vuser Scripts

Before recording a Baan Vuser script in VuGen, make sure that your machine can open a Baan session.

Follow these steps when creating a Baan Vuser script.

1 Create a Baan script in VuGen.

Create a new Baan template.

2 Record user actions.

Record typical user actions.

3 Add transactions, rendezvous, comments, and messages.

Use the Insert menu to add transactions, rendezvous, comments, and messages in order to enhance your script.

4 Add exception handling and set run-time properties.

Add functions to handle exceptions, set think time, and specify timeout periods. Configure run-time settings for logging and iterations.

5 Perform parameterization.

Replace recorded constants with parameters.

6 Save and run the Vuser script.

Run the Baan script from VuGen and view the Execution Log tab for run-time information.

Baan Vuser Functions

VuGen automatically records most of the functions listed in this section during a Baan user session. You can also manually program any of the functions into your script. For more information about the Baan Vuser functions, refer to the *Online Function Reference* (**Help > Function Reference**).

Session Functions

<code>init_session</code>	Initializes a Baan session.
<code>close_session</code>	Closes all Baan sessions and windows.
<code>start_session</code>	Begins a specific Baan session.
<code>set_exception</code>	Specifies how to handle exceptions.
<code>set_think_time</code>	Sets the think time range.
<code>set_default_timeout</code>	Sets the default timeout.

Button Object Functions

<code>button_press</code>	Activates a push button.
<code>button_set</code>	Sets the state of the specified radio button or check box.

Edit Object Functions

<code>edit_get_text</code>	Returns the text in an edit object.
<code>edit_set</code>	Replaces the entire contents of an edit object.
<code>edit_set_insert_pos</code>	Places the cursor at the specified point.
<code>edit_set_selection</code>	Selects text in an edit object.
<code>edit_type</code>	Types a string in an edit object.

List Object Functions

<code>list_activate_item</code>	Activates items in a list.
<code>list_select_item</code>	Selects a list item.
<code>list_get_selected</code>	Returns the currently selected item in a list.
<code>list_expand_item</code>	Shows hidden items in a list.
<code>list_collapse_item</code>	Hides items in a list.

Menu Object Functions

<code>menu_select_item</code>	Selects an item from a menu.
-------------------------------	------------------------------

Object Functions

<code>obj_get_info</code>	Returns the value of an object attribute.
<code>obj_get_text</code>	Reads text from an object.
<code>obj_mouse_click</code>	Clicks within an object.
<code>obj_mouse_dbl_click</code>	Double-clicks within an object.
<code>obj_mouse_drag</code>	Drags the mouse within an object.
<code>obj_type</code>	Sends keyboard input to an object.

Scroll Object Functions

<code>scroll_drag_from_min</code>	Drags a scroll object to the specified distance from the minimum position.
<code>scroll_line</code>	Scrolls the specified number of lines.
<code>scroll_page</code>	Moves a scroll object the specified number of pages.

Tab and Toolbar Object Functions

<code>tab_select_item</code>	Selects a tab in the active window.
<code>toolbar_button_press</code>	Clicks a toolbar button.

Static Object Functions

<code>static_get_text</code>	Returns the contents of a static text object.
------------------------------	---

Synchronization Functions

<code>obj_wait_info</code>	Waits for the value of an object attribute.
<code>tbl_wait_selected_cell</code>	Waits for the specified cell to appear in focus.
<code>win_wait_info</code>	Waits for the value of a window attribute.

Table Functions

<code>tbl_activate_cell</code>	Clicks Enter in the specified cell.
<code>tbl_get_cell_data</code>	Retrieves the contents of the specified cell from a table.
<code>tbl_get_selected_cell</code>	Returns the cell currently in focus in a table.
<code>tbl_press_zoom_button</code>	Clicks the table's zoom button.
<code>tbl_set_cell_data</code>	Sets the contents of a cell to the specified text in a table.
<code>tbl_set_selected_cell</code>	Selects a table cell.
<code>tbl_set_selected_rows</code>	Selects the specified rows in a table.

Window Object Functions

<code>set_window</code>	Specifies the window that receives subsequent input.
<code>win_activate</code>	Activates a window.
<code>win_close</code>	Closes a window.
<code>win_get_text</code>	Reads text from a window.
<code>win_get_info</code>	Returns the value of a window attribute.
<code>win_max</code>	Maximizes a window to fill the entire screen.
<code>win_min</code>	Minimizes a window to an icon.
<code>win_mouse_click</code>	Clicks within a window.
<code>win_mouse_dbl_click</code>	Double-clicks within a window.
<code>win_mouse_drag</code>	Drags the mouse within a window.
<code>win_move</code>	Moves a window to a new absolute location.
<code>win_resize</code>	Resizes a window.
<code>win_restore</code>	Restores a window from an iconized or maximized state to its previous size.
<code>win_type</code>	Sends keyboard input to a window.

Miscellaneous Functions


wait Causes test execution to pause for a specified amount of time.

You can further enhance your script with general Vuser functions such as **lr_output_message** and **lr_rendezvous**. For information on the Vuser functions, refer to the *Online Function Reference*.

Creating a Baan Vuser Script

After you create a Baan template, you begin recording user actions.

To create a new Baan Vuser script:

- 1 Select the *vuser_init* section, in order to record the login procedure into that section.
- 2  Click the **Record** button and specify the location of the Baan application in the Start Recording dialog box.
- 3 Switch to the *Actions* section and record typical user actions.
- 4 Insert Baan Vuser functions for think time, handling exceptions, and setting timeouts.

```
set_think_time(MINTHINK,MAXTHINK);
set_window ("Menu browser [User: bsp] [812]", 10);
menu_select_item ("File;Run Program...");
...
```

- 5 Add transactions to the script. Choose **Insert > Start Transaction** to specify the beginning of a transaction, and **Insert > End Transaction** to specify the end of a transaction.

```
lr_start_transaction("all_str_ses");
  button_press0 ("F1_OK");
  set_window ("tdpur4101m000: Maintain Purchase Orders [812]", 300);
lr_end_transaction("all_str_ses", LR_PASS);
```

- 6 Use the **Insert** menu to add rendezvous points, comments or messages to the script.
- 7 Parameterize your script. Click the string (in quotation marks) that you want to replace with a parameter, perform a right-click and choose **Replace with Parameter**. For more information see Chapter 7, “Defining Parameters.”
- 8 Set the appropriate run-time settings for iterations and logging.
- 9 Save the script and run it from VuGen.

Understanding Baan Vuser Scripts

The recorded script shows all the actions performed by the user during recording. The *Context Sensitive* functions show all the actions performed on the application’s objects. In the following example, VuGen recorded the focus to a window, the selection of a menu item, and the clicking of a button. In addition, a transaction was marked to analyze the time it takes for the object *Form1* to become in focus.

```
set_window ("tccom1501m000: Display Customers [550]", 30);
menu_select_item ("Edit;Find... Ctrl+F");
set_window ("Display Customers - Find", 300);
type ("100004");
lr_start_transaction("rses_find");
button_press0 ("F1_OK");
set_window ("tccom1501m000: Display Customers [550]", 30);
obj_wait_info("Form 1","focused","1",100);
lr_end_transaction("rses_find", LR_PASS);
```

You can add control flow logic to create loops within your script, instead of performing an iteration on the entire script.

```
for (loop = 0; loop < READLOOP; loop++)
{
    set_window ("tccom1501m000: Display Customers [550]", 30);
    menu_select_item ("Edit;Find... Ctrl+F");
    set_window ("Display Customers - Find", 300);
    type ("100004");
    lr_start_transaction("rses_find");
    button_press0 ("F1_OK");
    set_window ("tccom1501m000: Display Customers [550]", 30);
    obj_wait_info("Form 1","focused","1",100);
    lr_end_transaction("rses_find", LR_PASS);
    ...
}
```

Note that you may need to parameterize statements, in order to avoid duplicating data to a database. For more information, see Chapter 7, “Defining Parameters.”

Customizing Baan Vuser Scripts

You can view and edit your script from within VuGen at any time. You can use the Baan-specific functions to customize the script execution in the following areas:

- ▶ Think Time
- ▶ Handling Exceptions
- ▶ Setting Timeouts

Think Time

You can set the think time range for script execution. The think time emulates the work pattern of an actual user—the time the user pauses between actions. You set the beginning and end of a think time range using the **set_think_time** function. After each statement the Vuser pauses for the duration of the think time, a random value within the specified range.

If your desired think time range is constant throughout the script, you can define the beginning and end ranges as constants as shown in the example below.

In the following example, the think time range was set from 500 to 1000 milliseconds:

```
#define MINTHINK 500
#define MAXTHINK 1000

int LR_FUNC Actions(LR_PARAM p)
{
    set_think_time(MINTHINK,MAXTHINK);
    set_window ("Menu browser [User: bsp] [812]", 10);
    ...
}
```

Handling Exceptions

You can instruct a Baan Vuser how to handle exceptions that occur during replay, such as a message or error windows.

Using the **set_exception** function, you specify a function to be executed when the exception is encountered.

In the following example, the **set_exception** function instructs the Vuser to execute the *close* function when the Print Sales Invoices window opens. The *close* function is defined earlier in the script.

```
int close(char title[])
{
    win_close(title);
}
Actions()
{
    set_exception("ttstps0014: Print Sales Invoices",close);
    set_window ("Menu browser [User: bsp] [812]", 10);
    menu_select_item ("File;Run Program...");
    set_window ("tttsk2080m000:Run Program [812]", 10);
    type ("tdsls4101m000");
    ...;
}
```

Setting Timeouts

You can set the default timeout period for your functions. This timeout is applied to all functions using synchronization, such as **obj_wait_info**, **win_wait_info**, etc.

In functions containing a parameter specifying a timeout period (such as **set_window**), the specified timeout overrides the default timeout.

```
button_press ("F3_Continue");  
win_wait_info("ttstpspopen: Select Device [000]","displayed","0",10);
```


Part XI

Legacy Protocols

48

Introducing RTE Vuser Scripts

RTE Vusers operate terminal emulators in Windows environments. This chapter describes the development of Windows-based RTE Vuser scripts.

This chapter describes:

- ▶ Introducing RTE Vusers
- ▶ Understanding RTE Vuser Technology
- ▶ Getting Started with RTE Vuser Scripts
- ▶ Using TE Functions
- ▶ Mapping Terminal Keys to PC Keyboard Keys

The following information applies only to RTE (Windows) Vuser scripts.

About Developing RTE Vuser Scripts

RTE Vusers operate terminal emulators in order to load 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 the development of 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 51, "Synchronizing RTE Vuser Scripts."

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 RTE-Gate, a terminal emulator.



RTE-Gate works like a standard terminal emulator, supporting common protocols such as IBM 3270 & 5250, VT100, and VT220.

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 49, “Recording RTE Vuser Scripts.”

2 Enhance the script.

Enhance the Vuser script by inserting transactions, rendezvous points, synchronization functions, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

5 Run the script from VuGen.

Run the script from VuGen to verify that it runs correctly. View the standard output to verify that the program is communicating properly with the server.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you successfully create an RTE script, you integrate it into a session step. For more information on integrating scripts in a session step, refer to your *ProTune Console User's Guide*.

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. For syntax and examples of the TE functions, refer to the *Online Function Reference* (**Help > Function Reference**).

Terminal Emulator Connection Function

TE_connect	Connects the terminal emulator to the specified host.
-------------------	---

Text Retrieval Functions

TE_find_text	Searches for text in a designated area of the screen.
---------------------	---

TE_get_text_line	Reads text from a designated line on the screen.
-------------------------	--

TE_get_cursor_pos	Returns the current location of the cursor.
--------------------------	---

TE_get_line_attribute	Returns information about text formatting.
------------------------------	--

System Variable Functions

<code>TE_getvar</code>	Returns the value of an RTE system variable.
<code>TE_setvar</code>	Sets the value of an RTE system variable.

Error Code Functions

<code>TE_perror</code>	Prints an error code to the Output window.
<code>TE_spperror</code>	Translates an error code into a string.

Typing Functions

<code>TE_type</code>	Sends a formatted string to the client application.
<code>TE_typing_style</code>	Determines the way text is typed into the terminal emulator.

Synchronization Functions

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

The following TE functions can be parameterized: **TE_connect**, **TE_find_text**, **TE_get_text_line**, and **TE_type**. For details on parameterizing function in Vuser scripts, see Chapter 7, “Defining Parameters.”

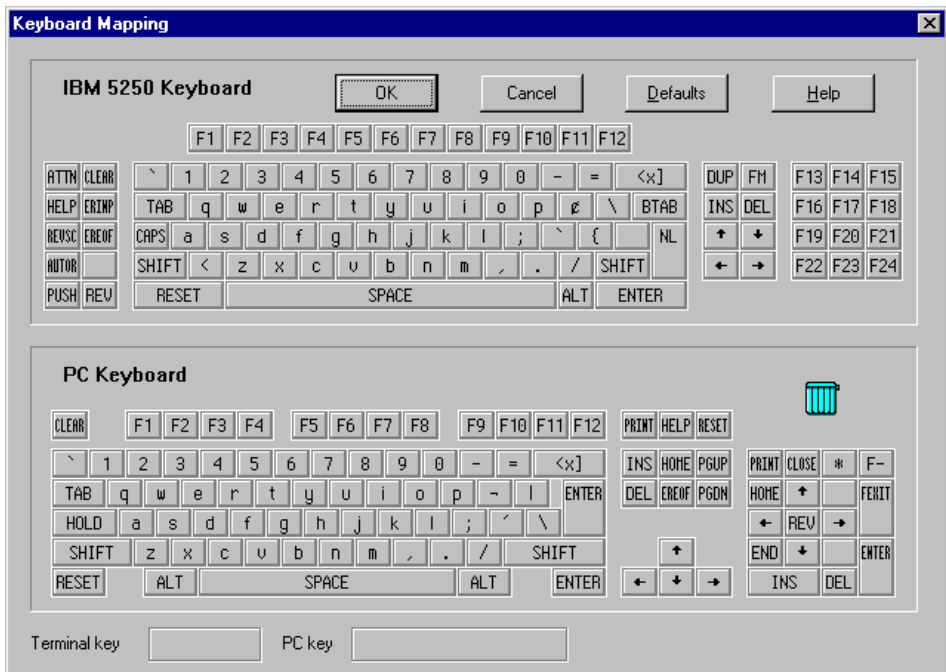
Mapping Terminal Keys to PC Keyboard Keys

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**.

49

Recording RTE Vuser Scripts

You use VuGen to record Windows-based Remote Terminal Emulation (RTE) Vuser scripts.

This chapter describes:

- ▶ Creating a New RTE Vuser Script
- ▶ Recording the Terminal Setup and Connection Procedure
- ▶ Recording Typical User Actions
- ▶ Recording the Log Off Procedure
- ▶ Setting the Recording Options
- ▶ Typing Input into a Terminal Emulator
- ▶ Generating Unique Device Names
- ▶ Setting the Field Demarcation Characters

The following information applies only to Terminal Emulation (RTE) Vuser scripts.

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.

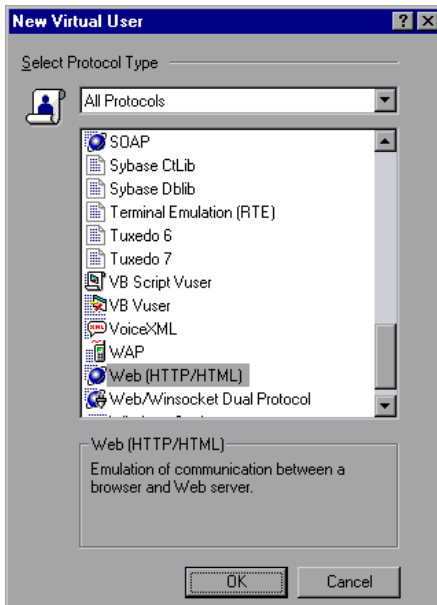
Before recording an RTE Vuser script, ensure that the recording options are set correctly. The recording options allow you to control how VuGen generates certain functions while you record a Vuser script. VuGen applies the recording options during all subsequent recording sessions.

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 Invoke VuGen. The VuGen window opens.
- 2 Click the **New** button. The **New Virtual User** dialog box opens:



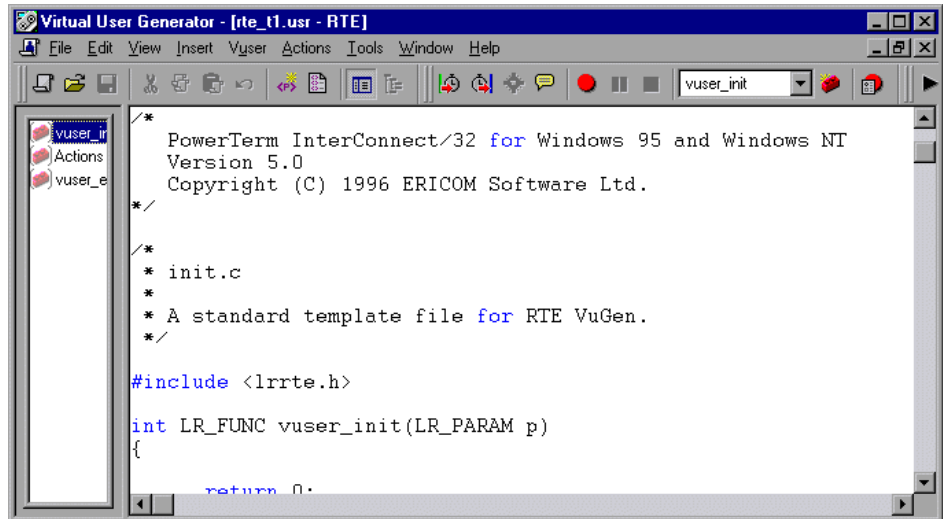
- 3 Select the **Legacy** protocol type, and choose **Terminal Emulator (RTE)**. Click **OK**. VuGen generates and displays an empty RTE script, with the cursor positioned to begin recording in the *vuser_init* section.

Recording the Terminal Setup and Connection Procedure

After you create a skeleton Vuser script, you record the terminal setup and connection procedure into the script. VuGen uses the PowerTerm terminal emulator when you record an RTE Vuser script.

To record the terminal setup and connection procedure:

- 1 Open an existing RTE Vuser script, or create a new one.
- 2 In the **Sections** box, select the section into which you want VuGen to insert the recorded statements. The available sections are *vuser_init*, *Actions*, and *vuser_end*.

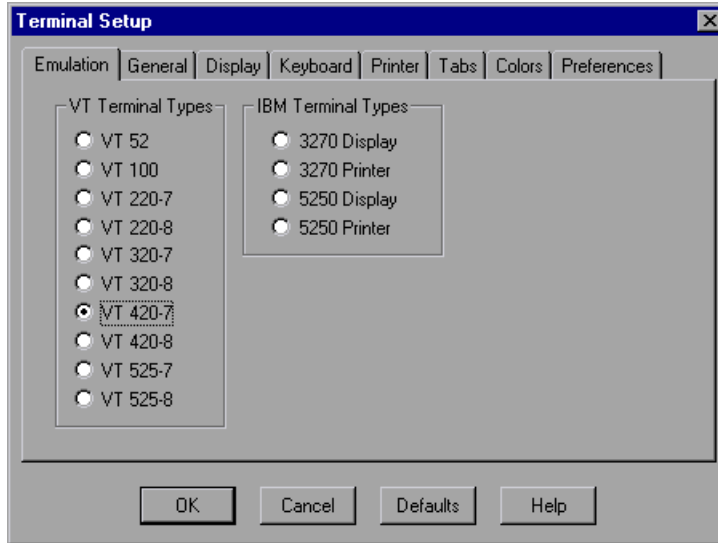


Note: Always record the terminal setup and connection procedure into the *vuser_init* section. The *vuser_init* section is not repeated when you run multiple iterations of a Vuser script—only the *Actions* section is repeated. For more information on the iteration settings, see Chapter 9, “Configuring Run-Time Settings.”

- 3 In the Vuser script, place the cursor at the location where you want to begin recording.



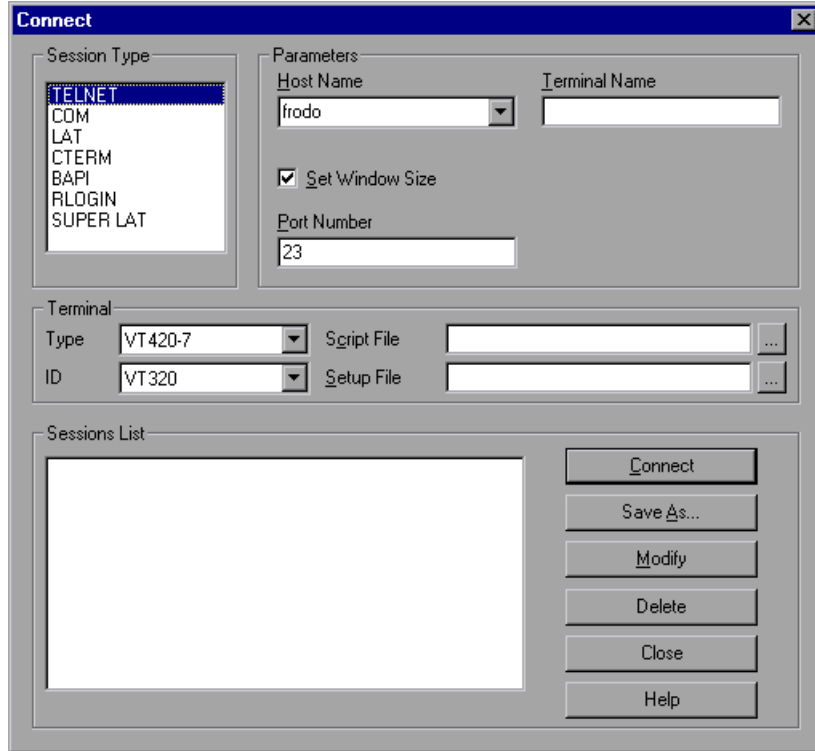
- 4 Click the **Record** button. The PowerTerm main window opens.
- 5 From the PowerTerm menu bar, select **Terminal > Setup** to display the Terminal Setup dialog box.



- 6 Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.

Note: Select an IBM terminal type to connect to an AS400 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**.

Note: You can save the parameters that you define for re-use in the future. To save the parameters, click **Save As**. 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 inserted **TE_connect** statement looks something like the following:

```
/* *** The terminal type is VT220-7. */
TE_connect("comm-type = telnet;host-name = frodo;
           terminal-type = vt220-7;terminal-model = vt320;", 60000);
if (TE_errno != TE_SUCCESS)
    return -1;
```

The inserted **TE_connect** statement is always followed by an *if* statement that checks whether or not the **TE_connect** function succeeds during replay.

Note: Do not record more than one connection to a server (**TE_connect**) in a Vuser script.

The terminal setup and connection procedure is complete. You are now ready to begin recording typical user actions into the Vuser script, as described below.

Recording Typical User Actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the *Actions* section of the Vuser script. Only the *Actions* section of a Vuser script is repeated when you run multiple iterations of the script. For details on setting iterations, see Chapter 9, “Configuring Run-Time Settings.”

When recording a session, VuGen records the text strokes and not the text. Therefore, it is not recommended that you copy and paste commands into the PowerTerm window—instead, type them in directly.

To record user actions:

- 1 Open an existing RTE Vuser script, and then click *Actions* in the **Section** box.
- 2 Proceed to perform typical user actions in the terminal emulator. VuGen generates the appropriate statements, and inserts them into the Vuser script while you type. If necessary, you can edit the recorded statements while you record the script.



Note: By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function. To change the waiting time, see “Setting the Recording Options,” on page 650.

When you finish recording the typical user actions, proceed to record the log off procedure, as described in the next section.

Recording the Log Off Procedure

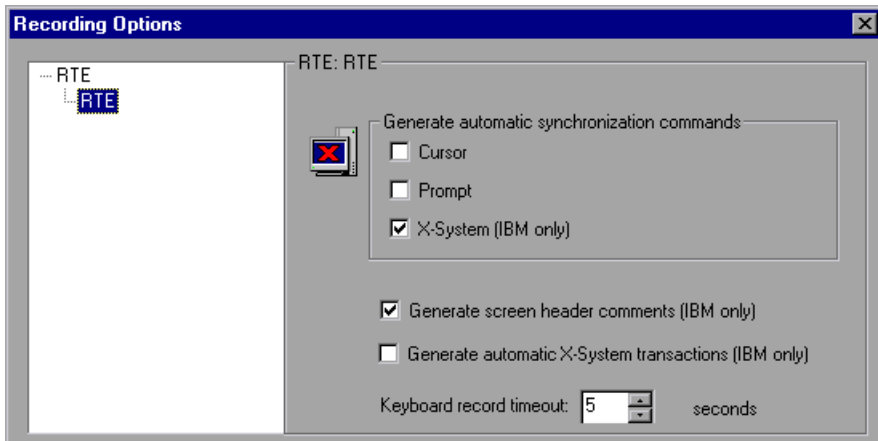
You record the Vuser log off process into the *vuser_end* section of the Vuser script. The *vuser_end* section is not repeated when you run many iterations of the script. For details on setting iterations, see Chapter 9, “Configuring Run-Time Settings.”

To record the log off procedure:

- 1 Ensure 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.

Setting the Recording Options

By setting the recording options, you can customize the code that VuGen generates for RTE functions. You use the Recording Options dialog box to set the recording options. To open the Recording Options dialog box, click the **Recording Options** button on the toolbar, or select **Tools > Recording Options**. Select the **RTE:RTE** node.



Automatic Synchronization Commands

VuGen can automatically generate a number of TE-synchronization functions, and insert them into the script while you record.

- 1 You can specify that VuGen generate a **TE_wait_sync** function each time a new screen is displayed while recording. To do so, select the “X-System” check box in the Recording Options dialog box.

By default, VuGen does automatically generate a **TE_wait_sync** function each time a new screen is displayed while recording.

Note: VuGen generates **TE_wait_sync** functions when recording IBM block mode terminals only.

- 2** You can specify that VuGen generate a **TE_wait_cursor** function before each **TE_type** function. To do so, select the **Cursor** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate **TE_wait_cursor** functions.

- 3** You can specify that VuGen generate a **TE_wait_text** function before each **TE_type** function (where appropriate). To do so, select the **Prompt** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate a **TE_wait_text** function before each **TE_type** function.

Note: VuGen generates meaningful **TE_wait_text** functions when recording VT type terminals only. Do not use automatic **TE_wait_text** function generation when recording block-mode (IBM) terminals.

Automatic Screen Header Comments (IBM terminals only)

You can instruct VuGen to automatically generate screen header comments while recording a Vuser script, and insert the comments into the script. Generated comments make a recorded script easier to read by identifying each new screen as it is displayed in the terminal emulator. A generated comment contains the text that appears on the first line of the terminal emulator window. The following generated comment shows that the Office Tasks screen was displayed in the terminal emulator:

```
/* OFCTSK                Office Tasks                */
```

To ensure that VuGen automatically generates comments while you record a script, select the “Generate screen header comments” check box in the Recording Options dialog box.

By default, VuGen does not automatically generate screen comments.

Note: You can generate comments automatically only when using block-mode terminal emulators such as the IBM 5250.

Automatic X-System Transactions (IBM terminals only)

You can specify that VuGen record the time that the system was in the X SYSTEM mode during a tuning session run. To do so, VuGen inserts a **TE_wait_sync_transaction** function after each **TE_wait_sync** function. Each **TE_wait_sync_transaction** function creates a transaction with the name “default”. Each **TE_wait_sync_transaction** function records the time that the system spent in the previous X SYSTEM state.

To instruct VuGen to insert **TE_wait_sync_transaction** statements while recording, select the “Generate automatic X SYSTEM transactions” check box in the Recording Options dialog box.

By default, VuGen does not automatically generate transactions.

Keyboard Recording Timeout

When you type text into a terminal emulator while recording, VuGen monitors the text input. After each keystroke, VuGen waits up to a specified amount of time for the next key stroke. If there is no subsequent keystroke within the specified time, VuGen assumes that the command is complete. VuGen then generates and inserts the appropriate **TE_type** function into the script.

To set the maximum amount of time that VuGen waits between successive keystrokes, enter an amount in the “Keyboard record timeout” box.

By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function.

Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to “type” character input into the RTE-GATE 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” below.
- **TE_typing_style** determines the speed at which the Vuser types. You can manually define the typing style by inserting a **TE_typing_style** function into the Vuser script. For details, see “Setting the Typing Style” below. Alternatively, you can set the typing style by using the run-time settings. For details, see “Configuring RTE Run-Time Settings,” on page 659.

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 function:

```
TE_type("<kReturn><kControl-y>");
```

depicts the input of the Return key followed by the Control and y keys. 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, refer to 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, refer to 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 “Configuring RTE Run-Time Settings,” on page 659.

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. For details about the run-time settings, see Chapter 9, “Configuring Run-Time Settings.”

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*.

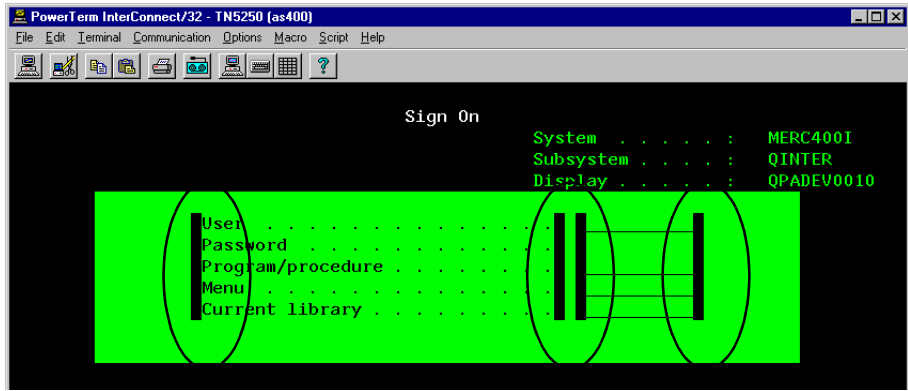
If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

In the following example, the **RteGenerateDeviceName** function generates unique device names with the format “TERMx”. The first name is TERM0, followed by TERM1, TERM2 etc.

```
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 choose to identify the character either as a space, or as 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.

50

Configuring RTE Run-Time Settings

After you record a Terminal Emulator script, you configure its run-time settings. This chapter describes the following Terminal Emulator Vuser run-time settings:

- Modifying Connection Attempts
- Specifying an Original Device Name
- Setting the Typing Delay
- Configuring the X-System Synchronization

The following information only applies to Terminal Emulator (TE) type Vusers.

About Terminal Emulator Run-Time Settings

After developing a Terminal Emulator Vuser script, you set the run-time settings. These settings let you control the behavior of the Vuser when running the script. Terminal Emulator run-time settings allow you to configure your TE Vusers so that they accurately emulate real users performing remote terminal emulation. You can configure settings for connection attempts, typing delay, and X-System synchronization.

You set the Terminal Emulator related run-time settings through the **RTE** node in the Run-Time Settings dialog box.



To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar. You can also modify the run-time settings from the ProTune Console.

This chapter only discusses the Run-Time settings for Terminal Emulator Vusers. For information about run-time settings that apply to all Vusers, see Chapter 9, “Configuring Run-Time Settings.”

Modifying Connection Attempts

The `TE_connect` function is generated by VuGen when you record a connection to a host. When you replay an RTE Vuser script, the `TE_connect` function connects the terminal emulator to the specified host. If the first attempt to connect is not successful, the Vuser retries a number of times to connect successfully. Details of each connection are recorded in the report file *output.txt*.

To set the maximum number of times that a Vuser will try to connect, enter a number in the **Maximum number of connection attempts** box in the RTE Run-Time settings.

By default, a Vuser will try to connect 5 times.

For more information about the `TE_connect` function, see the *Online Function Reference* (**Help > Function Reference**).

Specifying an Original Device Name

In certain environments, each session (Vuser) requires a unique device name. The `TE_connect` function generates a unique 8-character device name for each Vuser, and connects using this name. To connect using the device name (that is contained within the `com_string` parameter of the `TE_connect` function), select the **Use original device name** option in the RTE Run-Time settings.

Note: The original device name setting applies to IBM block-mode terminals only.

By default, Vusers use original device names to connect.

For details about the `TE_connect` function, see the *Online Function Reference* (**Help > Function Reference**).

Setting the Typing Delay

The delay setting determines how Vusers execute `TE_type` functions:

To specify the amount of time that a Vuser waits before entering the first character in a string, enter a value in the First key box, in milliseconds.

To specify the amount of time that a Vuser waits between submitting successive characters, enter a value in the Subsequent keys box, in milliseconds.

If you enter zero for both the first key and the subsequent key delays, the Vuser will send characters as a single string, with no delay between characters.

You can use the `TE_typing_style` function to override the Delay settings for a portion of a Vuser script.

For details about the `TE_type` and `TE_typing_style` functions, see the *Online Function Reference* (**Help > Function Reference**).

Configuring the X-System Synchronization

RTE Vuser scripts use the `TE_wait_sync` function for synchronization. You can set a timeout value and a stable-time value that VuGen applies to all `TE_wait_sync` functions. For details about the `TE_wait_sync` function, see the *Online Function Reference* (**Help > Function Reference**).

Timeout

When you replay a `TE_wait_sync` function, if the system does not stabilize before the synchronization timeout expires, the `TE_wait_sync` function returns an error code. To set the synchronization timeout, enter a value (in seconds) in the Timeout section of the RTE Run-Time settings.

The default timeout value is 60 seconds.

Stable Time

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X-SYSTEM mode. After the terminal returns from the X-SYSTEM mode, the Vuser still monitors the system for a short time. This ensures that the terminal has become stable, that is, that the system has not returned to the X-SYSTEM mode. Only then does the **TE_wait_sync** function terminate.

To set the time that a Vuser continues to monitor the system after the system has returned from the X-SYSTEM mode, enter a value (in milliseconds) in the Stable time box of the RTE Run-Time settings.

The default stable time is 1000 milliseconds.

51

Synchronizing RTE Vuser Scripts

Synchronization functions in an RTE Vuser script ensure that the input that a Vuser submits to a terminal emulator is synchronized with the responses from the server.

This chapter describes:

- ▶ Synchronizing Block-Mode (IBM) Terminals
- ▶ Synchronizing Character-Mode (VT) Terminals

The following information applies only to RTE (Windows) Vuser scripts.

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 669, 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 “MERCURY”. 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>MERCURY");
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 ensure 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** Choose **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, refer to 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("default");
```

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 tuning session 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** Choose **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, refer to 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. Refer to 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, refer to 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, refer to the *Online Function Reference* (**Help > Function Reference**).

52

Reading Text from the 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 describes:

- ▶ Searching for Text on the Screen
- ▶ Reading Text from the Screen

The following information applies only to RTE (Windows) Vuser scripts.

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

Part XII

Mailing Services Protocols

53

Developing Vuser Scripts for Mailing Services

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 describes:

- Getting Started with Mailing Services Vuser Scripts
- Working with IMAP Functions
- Working with MAPI Functions
- Working with POP3 Functions
- Working with SMTP Functions

The following information applies only to IMAP, MAPI, POP3, and SMTP Virtual User scripts.

About Developing Vuser Scripts for Mailing Services

The Mailing Service protocols emulate a user working with an email client, viewing and sending emails. The following mailing services are supported:

- Internet Messaging (IMAP)
- MS Exchange (MAPI)
- Post Office Protocol (POP3)
- Simple Mail Transfer Protocol (SMTP)

The mail protocols support both record and replay, with the exception of MAPI that only supports replay.

When you record an application using one of the mail protocols, VuGen generates functions that emulate the mail client's actions. You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see Chapter 4, “Selecting a Script Generation Language.” If the communication is performed through multiple protocols, you can record both of them. You can record several mail protocols, or a mail protocol together with HTTP or WinSock. For instructions on specifying multiple protocols, see Chapter 3, “Recording with VuGen.”

All Mailing Service functions come in pairs—one for global sessions and one where you can indicate a specific mail session. For example, **imap_logon** logs on to the IMAP server globally, while **imap_logon_ex** logs on to the IMAP server for a specific session.

Getting Started with Mailing Services Vuser Scripts

This section provides an overview of the process of developing Vuser scripts for Mailing Services using VuGen.

To develop a Mailing Service Vuser script:

1 Create a basic script using VuGen.

Invoke VuGen and create a new Vuser script for either a single mail protocol or multiple protocols.

2 Record the basic script using VuGen. (Except MAPI)

Choose an application to record. Perform typical operations in your application. For details, see Chapter 3, “Recording with VuGen.”

For MAPI, recording is not supported. Instead, you create an empty MAPI script and manually insert **mapi** functions into it. For examples, see the *Online Function Reference* (**Help > Function Reference**).

3 Enhance the script.

Enhance the script by inserting transactions and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

4 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

5 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, “Correlating Statements.”

6 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

7 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Virtual User script, you integrate it into a ProTune session. For more information on integrating Virtual User scripts in a ProTune session, refer to your *ProTune Console User's Guide*.

Working with IMAP Functions

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)*.

Function Name	Description
imap_append[_ex]	Appends a message to the end of a mailbox.

imap_check[_ex]	Requests a checkpoint for the current mailbox.
imap_close[_ex]	Closes the current mailbox.
imap_copy[_ex]	Copies mail messages to another mailbox.
imap_create[_ex]	Creates a mailbox.
imap_custom_request[_ex]	Executes a custom IMAP request.
imap_delete[_ex]	Deletes the specified mailbox.
imap_examine[_ex]	Examines a mailbox.
imap_expunge[_ex]	Removes all messages that are marked to be deleted.
imap_fetch[_ex]	Retrieves data associated with a mailbox message.
imap_free_ex	Frees an IMAP session descriptor.
imap_get_attribute_int[_ex]	Returns a mailbox attribute.
imap_get_attribute_sz[_ex]	Returns a mailbox attribute as a string.
imap_get_result[_ex]	Gets an IMAP server return code.
imap_list_mailboxes[_ex]	Lists the available mailboxes.
imap_list_subscriptions[_ex]	Lists the mailboxes that are subscribed or active.
imap_logon[_ex]	Logs in to an IMAP server.
imap_logout[_ex]	Logs off from an IMAP server.
imap_noop[_ex]	Performs a noop operation.
imap_search[_ex]	Searches a mailbox by keywords.
imap_select[_ex]	Selects a mailbox.
imap_status[_ex]	Requests the status of a mailbox.
imap_store[_ex]	Alters data associated with a mailbox message.

imap_subscribe[_ex]	Subscribes to or activates a mailbox.
imap_unsubscribe[_ex]	Unsubscribes from or deactivates a mailbox.

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.mycom-
              pany.com",
              LAST);

    imap_create("CreateMailboxes",
              "Mailbox=Products",
              "Mailbox=Solutions",
              "Mailbox=FAQs",
              LAST);

    imap_logout();

    return 1;
}

```

Working with MAPI Functions

MAPI Vuser script functions record activity to and from an MS Exchange server. Each MAPI function begins with an **mapi** prefix. For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

Function Name	Description
mapi_delete_mail[_ex]	Deletes the current or selected email entries.
mapi_get_property_sz[_ex]	Obtain a property value from the MAPI sessions.

- mapi_logon[_ex]** Logs on to MS Exchange.
- mapi_logout[_ex]** Logs out of MS Exchange.
- mapi_read_next_mail[_ex]** Reads the next mail in the mailbox.
- mapi_send_mail[_ex]** Sends an email to recipients.
- mapi_set_property_sz[_ex]** Sets a MAPI attribute.

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;
}
```

Working with POP3 Functions

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**).

Function Name	Description
<code>pop3_command[_ex]</code>	Sends a command to a POP3 server.
<code>pop3_delete[_ex]</code>	Deletes a message on the server.
<code>pop3_free[_ex]</code>	Frees the POP3 server from its commands.
<code>pop3_list[_ex]</code>	Lists the messages on the POP3 server.
<code>pop3_logoff[_ex]</code>	Logs off from a POP3 server.
<code>pop3_logon[_ex]</code>	Logs on to a POP3 server.
<code>pop3_retrieve[_ex]</code>	Retrieves messages from the POP3 server.

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;
}

```

Working with SMTP Functions

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)*.

Function Name	Description
smtp_abort_mail[_ex]	Aborts the transmission of an SMTP message.
smtp_free[_ex]	Frees the SMTP server from its commands.
smtp_logon[_ex]	Logs on to an SMTP server.
smtp_logout[_ex]	Logs off from an SMTP server.
smtp_send_mail[_ex]	Sends an SMTP message.
smtp_translate[_ex]	Translates an SMTP message.

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;
}

```


Part XIII

Middleware Protocols

54

Developing Jacada Vuser Scripts

VuGen allows you to record your communication with the Jacada Interface Server. You can run the recorded script or enhance it using standard Java library functions and VuGen-specific Java functions.

This chapter describes:

- ▶ Getting Started with Jacada Vusers
- ▶ Recording a Jacada Vuser
- ▶ Replaying a Jacada Vuser
- ▶ Understanding Jacada Vuser Scripts
- ▶ Working with Jacada Vuser Scripts

The following information only applies to Jacada Vuser scripts.

About Jacada Vuser Scripts

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 31, "Creating Web Vuser Scripts."

To create a script, you invoke VuGen and you record typical actions and business processes. VuGen generates a script that represents all of your actions. This script is java compatible.

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 ProTune session.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. Refer to Chapter 23, "Programming Java Scripts" for important information about function syntax and system configuration.

The next few sections discuss the recording options, run-time settings, and correlation.

Getting Started with Jacada Vusers

The following procedure outlines how to create Jacada Vuser scripts.

1 Ensure 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 23, "Programming Java Scripts" and the Read Me file.

2 Create a new Jacada Vuser script.

Select a *Jacada* type Vuser from the Middleware group.

3 Set the recording parameters and options for the Vuser script.

You specify the parameters for your applet or application such as working directory and paths. You can also set JVM, correlation, recorder, and debug recording options. For more information, see Chapter 13, "Setting Java Recording Options."

4 Record typical user actions.

Begin recording a script. Perform typical actions against your Jacada server. VuGen records your actions and generates a Vuser script.

5 Enhance the Vuser script.

Add VuGen functions to the Vuser script. For details, see Chapter 23, “Programming Java Scripts.” Use VuGen’s Function Navigator to add classes or methods. (see Chapter 45, “Performing EJB.”)

6 Parameterize the Vuser script.

Replace recorded constants with parameters. You can parameterize complete strings or parts of a string. For details, see Chapter 7, “Defining Parameters.”

7 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 15, “Configuring Java Run-Time Settings.”

8 Save and run the Vuser script.

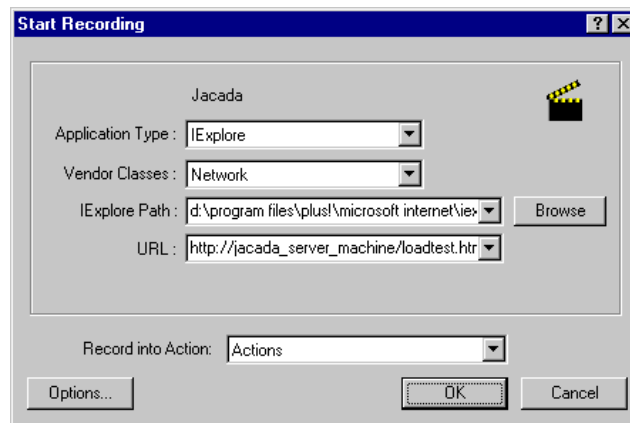
Run the script from VuGen and view the messages in the Execution log tab. For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

Recording a Jacada Vuser

You record a Jacada script to create a fully compatible Java program.

To record a Jacada script

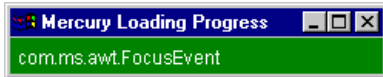
- 1 To begin recording, choose **File > New** and select *Jacada* from the Middleware Vuser type. The Start Recording dialog box opens.



- 2 Select an application type Internet Explorer or Netscape.
- 3 In the **Vendor Classes** box, the default is **Network** class. If *clbase.jar* is in your classpath, choose **Local** vendor classes.
- 4 Specify the browser path and the URL of the Jacada server start page.
 Note that a *Working Directory* is only necessary for applications that accesses a working directory (for example, reading property files or writing log files).
- 5 To set recording options, such as command line parameters for the JVM, click **Options**. For information about setting recording options, Chapter 13, “Setting Java Recording Options.”
- 6 In the **Record into Action** box, select the method into which you want to begin recording. The Actions class contains three methods: *vuser_init*, *action*, and *vuser_end*. The following table shows what to include into each method, and when each method is executed.

Script method	Used to emulate...	Is executed when...
<i>vuser_init</i>	a login to a server	the Vuser is initialized (loaded)
<i>action</i>	client activity	the Vuser is in “Running” status
<i>vuser_end</i>	a log off procedure	the Vuser finishes or is stopped

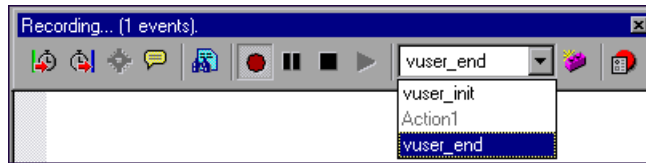
- 7 Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.



- 8 Perform typical actions within your application. Use the floating toolbar to switch methods during recording.



- 9 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.

- 10 Click **Stop Recording on** the Recording toolbar. The VuGen editor displays all the recorded statements.
- 11 Click **Save** to save the script. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name.

Replaying a Jacada Vuser

Ensure that you have properly installed a JDK version from Sun on the machine running ProTune—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.

Before replay, you must also download the *clbase.jar* file from the Jacada server. Add its location to the system classpath or to the **Additional Classpath** box in the Java VM tab of the Run-time Settings.

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 Mercury Interactive support.

For more information on the required environment settings, see Chapter 23, “Programming Java Scripts.”

Understanding Jacada Vuser Scripts

When you record a Jacada session, VuGen logs all calls to the server and generates a script with VuGen enhancements. These functions describe all of your actions within the application or applet. The script also contains exception handling for proper playback.

The recorded script is comprised of three sections:

► **Imports**

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script.

► **Code**

The **Code** section contains the Actions class and the recorded code within the *init*, *actions*, and *end* methods. The code section also contains the exceptions handler try-catch blocks for each command sent to the server.

► **Variables**

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 VuGen 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**).

Working with Jacada Vuser Scripts

The Actions method of a Jacada script, 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 VuGen Java functions. For a list of these functions and information on how to add them to your script, see Chapter 23, “Programming Java Scripts.”

55

Developing Tuxedo Vuser Scripts

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 describes:

- ▶ Getting Started with Tuxedo Vuser Scripts
- ▶ Using LRT Functions
- ▶ Understanding Tuxedo Vuser Scripts
- ▶ Viewing Tuxedo Buffer Data
- ▶ Defining Environment Settings for Tuxedo Vusers
- ▶ Debugging Tuxedo Applications
- ▶ Correlating Tuxedo Scripts

The following information applies only to PeopleSoft-Tuxedo and Tuxedo 6 and Tuxedo 7 Vuser scripts.

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 Tuxedo6 (for recording Tuxedo Version 6.x) or Tuxedo7 (for recording Tuxedo Version 7.x) as the type of Vuser. Choose an application to record. Record typical operations on your application.

For details, see Chapter 3, "Recording with VuGen."

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, “Correlating Statements.”

5 Configure the run-time settings.

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Tuxedo Vuser script, you integrate it into a ProTune session. For more information on integrating Vuser scripts in a ProTune session, refer to your *ProTune Console User's Guide*.

Using LRT Functions

The functions developed to emulate a Tuxedo client communications with a server are called LRT functions. Each LRT Vuser function has an **lrt** prefix. VuGen automatically records most of the LRT functions listed in this section during a Tuxedo session. You can also manually program any of the functions into your script. For syntax and examples of the LRT functions, refer to the *Online Function Reference* (**Help > Function Reference**).

Note: Some of the FML buffer functions indicate an optional “32” in the function name. These are the FML32 versions of the functions.

Buffer Manipulation Functions

<code>lrt_Fadd[32]_fld</code>	Adds a new field to an FML buffer.
<code>lrt_Finitialize[32]</code>	Initializes an existing FML buffer <i>fbfr</i> .
<code>lrt_Fldid[32]</code>	Maps a field name to a field identifier.
<code>lrt_Fname[32]</code>	Provides a map field identifier to field name.
<code>lrt_memcpy</code>	Copies the specified amount of bytes from the source to the destination.
<code>lrt_strcpy</code>	Copies a string like the C function <code>strcpy</code> .
<code>lrt_tmalloc</code>	Returns a pointer to a buffer type of <i>type</i> .
<code>lrt_tprealloc</code>	Changes the size of a typed buffer.
<code>lrt_tpfree</code>	Frees a typed buffer.
<code>lrt_tptypes</code>	Determines information about a typed buffer.

Client/Server Session Functions

<code>lrt_tpchkauth</code>	Checks if authentication is required by the application.
<code>lrt_tpinitialize</code>	Enables a client to join a System/T application.
<code>lrt_tpterm</code>	Removes a client from a System/T application.

Communication Functions

<code>lrt_tpacall</code>	Sends a service request.
<code>lrt_tpbroadcast</code>	Broadcasts notification by name.
<code>lrt_tpcall</code>	Sends a service request and awaits its reply.

<code>lrt_tpcancel</code>	Cancels a call descriptor.
<code>lrt_tpchkunsol</code>	Checks for an unsolicited message.
<code>lrt_tpconnect</code>	Establishes a conversational service connection.
<code>lrt_tpdequeue</code>	Dequeues a message from a queue.
<code>lrt_tpdison</code>	Terminates a conversational service connection.
<code>lrt_tpenqueue</code>	Stores a message in the queue.
<code>lrt_tpgetrply</code>	Returns a reply from a previously sent request.
<code>lrt_tpgprio</code>	Returns the priority for the last request sent or received.
<code>lrt_tpnotify</code>	Sends notification to a client.
<code>lrt_tppost</code>	Posts an event.
<code>lrt_tprecv</code>	Receives a message in a conversational connection.
<code>lrt_tpsend</code>	Sends a message in a conversational connection.
<code>lrt_tpsetunsol</code>	Sets the method for handling unsolicited messages.
<code>lrt_tpsprio</code>	Sets the priority for the next request sent or forwarded.
<code>lrt_tpsubscribe</code>	Subscribes to an event.
<code>lrt_tpunsubscribe</code>	Unsubscribes to an event.

Environment Variable Functions

<code>lrt_set_env_list</code>	Sets a list of environment variables.
<code>lrt_tuxgetenv</code>	Returns a value corresponding to an environment name.

<code>lrt_tuxputenv</code>	Modifies an existing environment value or adds a value to the environment.
<code>lrt_tuxreadenv</code>	Adds variables to the environment from a file.

Error Processing Functions

<code>lrt_abort_on_error</code>	Aborts the current transaction, if the previous Tuxedo function call resulted in an error.
<code>lrt_fstrerror[32]</code>	Retrieves error message string for FML error.
<code>lrt_getFerror[32]</code>	Retrieves the error status code for the last FML operation that failed.
<code>lrt_gettperrno</code>	Retrieves the error status code for the last Tuxedo transaction monitor function.
<code>lrt_gettpurcode</code>	Retrieves the application return code.
<code>lrt_tpstrerror</code>	Retrieves error message string for System/T error.

Transaction Handling Functions

<code>lrt_tpabort</code>	Aborts the current transaction.
<code>lrt_tpbegin</code>	Begins a transaction.
<code>lrt_tpcommit</code>	Commits the current transaction.
<code>lrt_tpgetlev</code>	Checks if a transaction is in progress.
<code>lrt_tpresume</code>	Resumes a global transaction.
<code>lrt_tpscmt</code>	Sets when <code>lrt_tpcommit</code> should return.
<code>lrt_tpsuspend</code>	Suspends a global transaction.
<code>lrt_tx_begin</code>	Begins a global transaction.
<code>lrt_tx_close</code>	Closes a set of resource managers.
<code>lrt_tx_commit</code>	Commits a global transaction.

lrt_tx_info	Returns global transaction information.
lrt_tx_open	Opens a set of resource managers.
lrt_tx_rollback	Rolls back a global transaction.
lrt_tx_set_commit_return	Sets the <code>commit_return</code> characteristic to the value specified in <code>when_return</code> .
lrt_tx_set_transaction_control	Sets the <i>transaction_control</i> characteristic to the value specified in <code>control</code> .
lrt_tx_set_transaction_timeout	Sets the <code>transaction_timeout</code> characteristic to the value specified in <code>timeout</code> .

Correlating Statement Functions

lrt_display_buffer	Stores buffer information in a file.
lrt_save[32]_fld_val	Saves the current value of an FML buffer to a parameter.
lrt_save_parm	Saves a portion of a character array (such as a <code>STRING</code> or <code>CARRAY</code> 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.

Note: In general, it is recommended to 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, it is recommended to 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.

Understanding Tuxedo Vuser Scripts

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);
tprresult_int = lrt_tpbegin(30, 0);
data_0 = lrt_tmalloc("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=1 23-456-7890",
LRT_END_OF_PARMS);

lrt_fadd fld((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
New York, NY 10000", LRT_END_OF_PARMS);
lrt_fadd fld((FBFR*)data_0, "name=SSN", "value=111111111",
LRT_END_OF_PARMS);

lrt_fadd fld((FBFR*)data_0, "name=LAST_NAME",
"value=Doe", LRT_END_OF_PARMS);

lrt_fadd fld((FBFR*)data_0, "name=FIRST_NAME",
"value=Bj", LRT_END_OF_PARMS);

lrt_fadd fld((FBFR*)data_0, "name=SAMOUNT",
"value=0.00", LRT_END_OF_PARMS);
```



```

/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, &data_0, &olen_2, 0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();

```

Using Parameters in Tuxedo Scripts

You can define parameters in Tuxedo scripts, as described in Chapter 7, “Defining Parameters.” Note that Tuxedo scripts contain strings of type “name=...” or “value=...”. You can only define parameters for the portion of the string following the equal sign (=). For example:

```

lrt_Fadd fld((FBFR*)data_0,"name=PHONE", "value=<parameter_1>",
            LRT_END_OF_PARMS);

```

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,” on page 706. 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.

The screenshot shows the Virtual User Generator (VuGen) interface. The title bar reads "Virtual User Generator - [tux63_tst1.usr - TUXEDO63]". The menu bar includes File, Edit, View, Insert, Vuser, Actions, Tools, Window, and Help. The toolbar contains various icons for file operations and execution. The left pane shows a tree view with the following items: vuser_init, Actions, vuser_end, and replay.vdf. The main editor window displays the content of the replay.vdf file, which is a C program generated by LoadRunner. The code includes preprocessor directives, a character array declaration, and comments describing the data fields.

```

/* This file is generated by LoadRunner. You may edit it
#ifndef TUXVDF_H
#define TUXVDF_H
#define binDat_1 \
    ""
char* data_0;
/* Returned FML buffer 1
field: "name=ACCOUNT_ID", "occurrence=0", "value=10002"
field: "name=FORMNAM", "occurrence=0", "value=CBALANCE"
field: "name=SBALANCE", "occurrence=0", "value=$346.00"
Reply buffer */
#endif /* TUXVDF_H */

```

Defining Environment Settings for Tuxedo Vusers

The following section describes the system variable settings for Tuxedo Vusers running on Windows and UNIX platforms. You define the system variables in your Control Panel/System dialog box (NT) or .cshrc or .login file (UNIX).

TUXDIR	the root directory for Tuxedo sources.
FLDTBLDIR	list of directories containing FML buffer information. In Windows, separate the names of directories with semi-colons. On UNIX platforms, separate the names of the directories with a colon.
FIELDTBLS	list of files containing FML buffer information. On both Windows and UNIX platforms, separate the file names with commas.

For example:

```
SET FLDTBLDIR=%TUXDIR%\udataobj;%TUXDIR%\APPS\WS (PC)
SET FIELDTBLS=bankflds,usysflds (PC)
setenv FLDTBLDIR $TUXDIR/udataobj:$TUXDIR/apps/bankapp (Unix)
setenv FIELDTBLS bank.flds,Usysflds (Unix)
```

You must define the following system variables for Tuxedo clients using Tuxedo/WS workstation extensions during execution:

WSNADDR	specifies the network address of the workstation listener process. This enables the client application to access Tuxedo. Note that to define multiple addresses in a WSNADDR statement, each address must be separated by a comma.
WSDEVICE	specifies the device that accesses the network. Note that you do not need to define this variable for some network protocols.

For example:

```
SET WSNADDR=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 7** to record applications using Tuxedo 7.1.

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, refer to 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 angle 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, it is recommended to 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, it is recommended to 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.

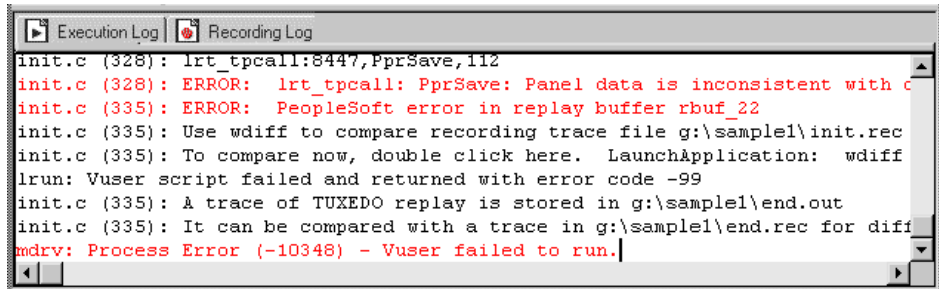
Determining Which Values to Correlate

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 execution log.

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



```

Execution 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 Chapter 8, “Correlating Statements.”.

To correlate statements using `lrt_save_parm`:

Once you decide which value to correlate, you can use `lrt_save_parm` to save a portion of a character array (such as a `STRING` or `CARRAY` buffer) to a parameter.

- 1 Insert the `lrt_save_parm` statement in your script at the point where you want to save the contents of the current buffer.

```
lrt_save_parm (buffer, offset, length, "param_name");
```

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

View the buffer contents by selecting the `replay.vdf` file in the Data Files box of the main VuGen window.

- 3 Replace all instances of the value with the parameter name in angle 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
\x0"
"X"
"\x89\x0\x0\x0\xb\x0"
"SPprLoadReq"
"\xff\x0\x10\x0\x0\x4\x3\x6"
"<empid>" // G001
"\x7"
"Claudia"
"\xe"
"LAST_NAME_SRCH"
...
```

This function can also be used to save a portion of a character array within an FML buffer. In the following example, the phone number is a character array, and the area code is the first three characters. First, the

`lrt_save fld_val` statement saves the phone number to a parameter, *phone_num*. The `lrt_save_parm` statement uses `lr_eval_string` to turn the phone number into a character array and then saves the area code into a parameter called *area_code*.

```
lrt_save fld_val((FBFR*)data_0, "name=PHONE", 0, "phone_num");
lrt_save_parm(lr_eval_string("<phone_num>"), 0, 3, "area_code");
lr_log_message("The area code is %s\n", lr_eval_string("<area_code>"));
```

To correlate statements using `lrt_save_searched_string`:

Use `lrt_save_searched_string` to search for a string in a buffer, and save a portion of the buffer, relative to the string occurrence, to a parameter.

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

```
lrt_save_searched_string (buffer, buf_size, occurrence, string, offset,
                        length, "param_name");
```

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

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

View the buffer contents by selecting the *replay.vdf* file in the Data Files box of the main VuGen window.

- 3 Replace all instances of the value with the parameter name in angle 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 always 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,
    TPSIGRSTR);

/* 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");
```

Part XIV

Streaming Data Protocols

56

Developing Streaming Data Vuser Scripts

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 and Media Player are applications that display streaming content.

You use VuGen to record communication between a client application and a server that communicate using the RealPlayer or Media Player protocol. The resulting script is called a Real or Media Player Vuser script.

This chapter describes:

- ▶ Getting Started with Streaming Data Vuser Scripts
- ▶ Using RealPlayer LREAL Functions
- ▶ Using Media Player MMS Functions

The following information applies only to Real and Media Player Protocols.

About Recording Streaming Data Virtual User Scripts

The Streaming Data protocols allows you to emulate a user playing media or streaming data files.

When you record an application using a streaming data protocol, VuGen generates functions that describe your actions. For RealPlayer sessions, VuGen generates functions with an **lreal** prefix. For Media Player sessions,

VuGen uses functions with an **mms** prefix. Note that recording is not supported for Media Player mms functions—only replay.

Getting Started with Streaming Data Vuser Scripts

This section provides an overview of the process of developing RealPlayer and Media Player streaming data Vuser scripts using VuGen.

To develop a RealPlayer or Media Player Vuser script:

1 Record the basic script using VuGen. (RealPlayer only)

For RealPlayer, invoke VuGen and create a new Virtual Playerscript. Choose an application to record. Record typical operations on your application. For details, see Chapter 3, “Recording with VuGen.”

For Media Player, recording is not supported. Instead, you create an empty Media Player script and manually insert **mms** functions into it. For examples, see the *Online Function Reference* (**Help > Function Reference**).

2 Enhance the script.

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

3 Define parameters (optional).

Define parameters for the fixed-values recorded into your script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

4 Correlate statements (optional).

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 8, “Correlating Statements.”

5 Configure the run-time settings.

The run-time settings control the Virtual Player/User behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 9, “Configuring Run-Time Settings.”

6 Run the script from VuGen.

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Vuser script, you integrate it into a ProTune session. For more information on integrating Vuser scripts in a ProTune session, refer to your *ProTune Console User’s Guide*.

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

lreal_clip_size	Returns the size of the current clip.
lreal_close_player	Closes a RealPlayer instance.
lreal_open_player	Creates a new RealPlayer instance.
lreal_open_url	Opens a URL.
lreal_pause	Pauses the playing of a RealPlayer clip.
lreal_play	Plays a RealPlayer clip.
lreal_seek	Seeks a position in a RealPlayer clip.
lreal_stop	Stops playing a RealPlayer clip.

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.

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.

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.

For detailed syntax information on these functions, see the *Online Function Reference* (**Help > Function Reference**).

Function Name	Description
mms_close[_ex]	Closes the Media Player.
mms_get_property[_ex]	Retrieves a property of a Media Player clip.
mms_isactive[_ex]	Verifies that the Media Player is active.
mms_pause[_ex]	Pauses the playing of a Media Player clip.
mms_play[_ex]	Plays a Media Player clip.
mms_resume[_ex]	Resumes playing a Media Player clip.
mms_sampling[_ex]	Samples a Media Player clip.
mms_set_property[_ex]	Sets a Media Player clip property.
mms_set_timeout[_ex]	Sets a timeout value for a Media Player clip.
mms_stop[_ex]	Stops playing a Media Player clip.

For example, the **mms_play** function takes the 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);
```


Part XV

Wireless Protocols

57

Introducing Wireless Vusers

You use VuGen to develop scripts for wireless applications using the WAP, VoiceXML, or i-mode protocols. VuGen creates Vuser scripts by recording your actions over a wireless network.

This chapter describes:

- Understanding the WAP Protocol
- Understanding the i-mode System
- i-mode versus WAP
- Understanding VoiceXML

About Wireless Vusers

VuGen supports two wireless protocols:

- WAP (Wireless Application Protocol)
- i-mode
- VoiceXML

Each protocol has specific characteristics, differing in both the implementation and development of user content.

Developers use toolkits that serve as a development environment for creating content and applications for the wireless protocols.

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:

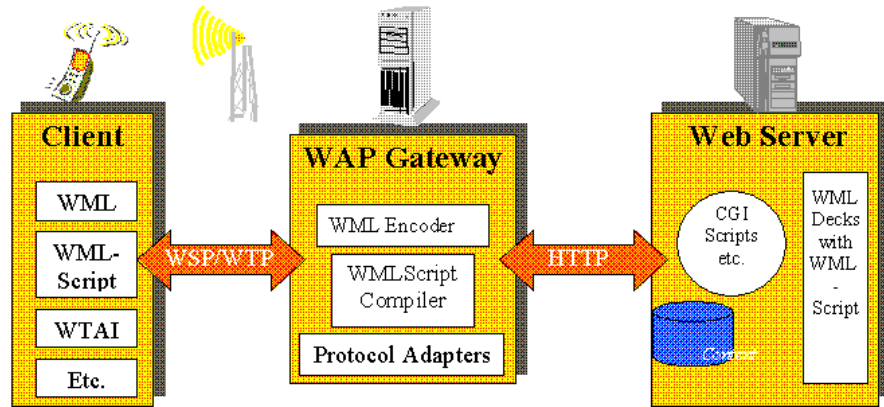
- ▶ acts as a gateway between the wireless network and the wire-line Internet
- ▶ provides protocol translation
- ▶ optimizes data transfer for the wireless handset

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 lets your record in two modes: WSP and 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 735.

Understanding the i-mode System

The i-mode protocol is NTT DoCoMo's mobile Internet access system. Technically, i-mode is an overlay over ordinary mobile voice systems. While the voice systems are *circuit-switched* (that is, you need to dial-up) i-mode is *packet-switched*. This means that i-mode is in principle always connected, provided the i-mode signal can reach you. When you select an i-mode item on the handset menu, the data is usually downloaded immediately, without the usual dial-up delay. However, there may be a delay in receiving the data, depending on the size of the data and network bandwidth.

Working with i-mode is similar to accessing the Internet with a browser. For example, they send e-mail, look at the weather forecasts, sports results, play games, execute online stock trades, purchase air tickets, and search for restaurants.

The i-mode protocol uses *cHTML* (compact HTML), a subset of ordinary HTML. In addition to standard HTML tags, there are several i-mode specific tags. For example, one i-mode tag sets up a link, which dials up to a telephone number. Another i-mode-specific tag informs search engines that a particular web page is an i-mode page.

In addition, there are many DoCoMo special characters which serve as symbols. For example, there are special characters that represent joy, love, sadness, telephone, trains, encircled numbers, and so forth.

Since cHTML is a subset of HTML, you can use your Netscape or IE browser to view i-mode pages, such as <http://www.eurotechnology.com/i/> or <http://www.eu-japan.com/i/>. However, since nearly all i-mode users are Japanese, almost all i-mode content is in the Japanese language. Therefore, you will need Japanese Text Display support in your browser. When you view i-mode content in a regular browser, you will not be able to see i-mode-specific tags. In addition, you cannot display the special DoCoMo-i-mode symbols.

i-mode Toolkits

To assist developers in producing i-mode services, several toolkits are available and supported by VuGen. The i-mode toolkits provide an environment for developers who want to provide Internet services and content for mobile terminals. Toolkits allow developers to write, test, debug,

and run applications on a PC-based simulator phone. The toolkit allows users to browse i-mode sites through a standard HTTP connection. A partial list of the supported toolkits are CompactViewer, and Pixa versions 2.0 and 2.1.

i-mode versus WAP

There are several important differences in the way i-mode and WAP based services are presently implemented. i-mode uses cHTML, a subset of HTML which is relatively easier to master than WAP's markup language WML. Currently, i-mode is implemented with a packet-switched system, which is in principle "always on" while WAP systems use a circuit-switched model, that is, dial-up. Note that packet-switching or circuit-switching is a technical difference of the telecommunication system on which the services are based. In principle, i-mode and WAP encoded web pages can be delivered over packet or circuit switched systems.

An additional difference is in the pricing methods: an i-mode user is charged for the amount of information downloaded, plus various premium service charges. WAP users are charged by the connection time.

Understanding VoiceXML

VoiceXML or VXML, is a technology that allows you to interact with the Internet using voice-recognition technology through a voice browser or a telephone. Using VoiceXML, you interact with voice browser by listening to pre-recorded or computer-synthesized audio and submitting input through a natural speaking voice or a keypad, such as a telephone.

A VoiceXML consists of a VoiceXML gateway that accesses static or dynamic VoiceXML content on the Web. The gateway has a VoiceXML browser, Text-To-Speech, Automatic Speech Recognition (ASR), and the telephony hardware that connects to a Public Switched Telephone Network (PSTN). It connects to the phone network through one of the following lines: T1, POTS, or ISDN. A Plain Old Telephone Server (POTS) line, similar to the ones used in residential locations, only handles a single connection; a T1 line has 24 individual phone lines.

A typical voice dialog consists of:

- 1** You dial up the system by phone (wireless or fixed). The telephony hardware picks it up and passes the call to the VoiceXML browser.
- 2** The VoiceXML gateway retrieves a VoiceXML document with a *vxml* extension from the specified Web server, and plays a prompt tone.
- 3** You speak into the telephone or press a key on the phone keypad.
- 4** The telephony equipment passes the recorded sound to the speech recognition engine (if it's speech), using a predefined dictionary contained in the VoiceXML document.
- 5** The VoiceXML browser executes the commands in the document based upon the results of the speech analysis, and plays another pre-recorded or synthesized prompt.

VuGen supports recording for VoiceXML sessions. The recorded script contains **web_url** functions that emulate your actions. In the following example, a user requests the a page with stock information.

```
Action1()
{
    web_add_auto_header("Accept",
        "text/x-xml, */*");

    web_add_auto_header("Content-Type",
        "application/x-www-form-urlencoded");

    web_add_auto_header("User-Agent",
        "Motorola VoxGateway/2.0");

    web_url("top.vxml",
        "URL=http://testserver1/Vxmlexample/top.vxml?DNIS=-",
        "Resource=0",
        "RecContentType=application/octet-stream",
        "Referer=",
        "Mode=HTTP",
        LAST);

    web_url("stock.vxml",
        "URL=http://testserver1/Vxmlexample/stock.vxml",
        "Resource=0",
        "RecContentType=application/octet-stream",
        "Referer=",
        "Mode=HTTP",
        LAST);
    return 0;
}
```


58

Recording Wireless Vuser Scripts

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 describes:

- Getting Started with Wireless Vuser Scripts
- Using Wireless Vuser Functions
- Troubleshooting Wireless Vuser Scripts

The following information only applies to all Wireless protocols, WAP, i-mode, and VoiceXML.

About Recording Wireless Vuser Scripts

Suppose you have a Web site that displays purchase request status by customers. You want to ensure 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 session step, in which the Web or WAP server services the simultaneous requests for information. Each Vuser could:

- load an opening page
- submit a request
- 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.

Getting Started with Wireless Vuser Scripts

This section provides an overview of the process of developing Wireless Vuser scripts using VuGen.

To develop a Wireless script:

1 Create a new script using VuGen.



Select **File > New** or click the **New** button to create a new script in either single or multiple protocol mode.

For details about creating a new script, see Chapter 3, “Recording with VuGen.”

2 Set the recording options.

Set the recording options. For information about setting common Internet recording options, see Chapter 34, “Setting Recording Options for Internet Protocols.” For information about Wireless specific recording options, see Chapter 60, “Setting Wireless Vuser Recording Options.”

3 Record the actions using VuGen.

Record the actions over the toolkit session.

For information about recording, see Chapter 3, “Recording with VuGen.”

4 Enhance the Vuser script.

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 6, “Enhancing Vuser Scripts.”

5 Define parameters (optional).

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 7, “Defining Parameters.”

6 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, and performance preferences.

For information about the General run-time settings, see Chapter 9, “Configuring Run-Time Settings.”

For information about common Internet protocol run-time settings, see Chapter 36, “Configuring Internet Run-Time Settings.”

For information about WAP specific run-time settings, see Chapter 61, “Configuring WAP Run-Time Settings.”

7 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 539.

8 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 stand-alone test, see Chapter 10, “Running Vuser Scripts in Stand-Alone Mode.”

After you create a Wireless Vuser script, you integrate it into a session step. For more information, refer to the *ProTune Console User's Guide*.

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 message functions and custom C functions to your Vuser scripts after recording.

The functions representing standard HTTP actions, have a **web** prefix. For information about these functions, see Chapter 32, “Using Web Vuser Functions.”

General Vuser functions begin with an **lr** prefix. For more information, see “Using C Vuser Functions,” on page 18.

The following section describes the functions representing WAP specific actions, which have a **wap** prefix.

For a complete list of all Web related functions, see Chapter 32, “Using Web Vuser Functions” or refer to the *Online Function Reference* (**Help > Function Reference**).

For WAP Vusers running scripts in Wireless Session Protocol (WSP) mode, only the following functions are supported.

Action Functions:	web_custom_request , web_submit_data , and web_url
Authentication Functions:	All— web_set_user , web_set_certificate[_ex]
Cookie Functions:	All— web_add_cookie , web_cleanup_cookie , web_remove_cookie
Header Functions:	All — web_add_auto_header , web_add_header , web_cleanup_auto_headers , web_save_header
Correlation Functions:	All— web_create_html_param[_ex] , web_reg_save_param , web_set_max_html_param_len

The following section lists the WAP specific functions:

WAP Specific Functions

<code>wap_add_const_header</code>	Specifies a constant header to pass to a WAP gateway.
<code>wap_connect</code>	Connects to a WAP gateway.
<code>wap_disconnect</code>	Disconnects from a WAP gateway.
<code>wap_format_si_message</code>	Formats an SI type message
<code>wap_format_sl_message</code>	Formats an SL type message
<code>wap_mms_msg_add_field</code>	Adds a field to an MMSC message.
<code>wap_mms_msg_add_multipart_entry</code>	Adds a multipart entry to an MMSC message.
<code>wap_mms_msg_create</code>	Creates a message for an MMSC.
<code>wap_mms_msg_destroy</code>	Destroys an MMSC message
<code>wap_mms_msg_submit</code>	Sends a message to the MMSC
<code>wap_pi_push_cancel</code>	Cancels a message sent to a PPG.
<code>wap_pi_push_submit</code>	Submits a Push message.
<code>wap_radius_connection</code>	Connects or disconnects from a RADIUS server.
<code>wap_send_sms</code>	Sends an SMS type message.
<code>wap_set_bearer</code>	Sets the underlying bearer-UDP or CIMD2 (SMS).
<code>wap_set_capability</code>	Sets a client capability for a WAP gateway connection.
<code>wap_set_connection_mode</code>	Sets the connection mode and security level.
<code>wap_set_gateway</code>	Sets a gateway IP address and port.
<code>wap_set_sms_user</code>	Sets login information for the SMSC.
<code>wap_wait_for_push</code>	Waits for a Push message to arrive.

For more information, select the function in the VuGen editor and press F1, or refer to the *Online Function Reference* (**Help > Function Reference**).

Troubleshooting Wireless Vuser Scripts

Nokia Toolkits

For Nokia toolkits (1.3 and 2.0), you need to launch the toolkit manually assigning it the proper IP address.

To relaunch the toolkit:

- 1** Ensure that there is no gateway running on the VuGen machine. The presence of another gateway could block the port for the pseudo-gateway.
- 2** Start VuGen.
- 3** Invoke the toolkit.
- 4** In the Recording Options, choose the **Internet Protocol:WAP Toolkit** node and select **Manually launch a WAP toolkit**.
- 5** Click **OK** and begin recording.
- 6** Select the **Internet Protocol:Recording Mode** node and copy the Gateway IP assigned by VuGen. Paste this IP address into the toolkit's connection settings.
- 7** In the toolkit, enter the desired URL. Ignore the message *Gateway not connected*. Enter the URL again. VuGen may have recorded several **web_add_const_header** events at this point.

Every new recording session requires closing the toolkit and repeating steps 2 - 7.

Note: You can use the above procedure for other toolkits for which you encounter recording issues.

59

Working with WAP Vuser Scripts

You use VuGen to develop WAP (Wireless Application Protocol) Vuser scripts. VuGen creates Vuser scripts by recording your actions while you operate a WAP device.

This chapter describes:

- Recording Over a Phone
- Bearers Support
- RADIUS Support
- Push Support
- ProTune Push Support
- MMS Support

About WAP Vusers

The Wireless Application Protocol (WAP) is an open specification that enables mobile users with wireless devices to access and interact with information and services instantly. For an overview of WAP technology, see Chapter 57, “Introducing Wireless Vusers.”

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 735.

You can record a WAP session using a toolkit or through your phone. For information about recording through a toolkit, see Chapter 60, “Setting

Wireless Vuser Recording Options.” For information about recording over a phone, see the following section.

You can program scripts to emulate WAP sessions using the **wap** Vuser functions. For more information and examples, see the *Online Function Reference (Help > Function Reference)*.

VuGen support for WAP allows you to choose a bearer, identify a RADIUS server, and emulate a Push mechanism. This support is described in this chapter.

Recording Over a Phone

You can record a WSP session between phones or toolkits and a WAP gateway. In order to record the WSP session, make sure that the toolkit or phone gateways settings are configurable.

During recording, VuGen launches a pseudo gateway. VuGen captures the WSP traffic on this gateway and creates a script.

To configure VuGen for a WSP recording session, you must enable WSP in the **Recording Mode** tab of the recording options (see Chapter 60, “Setting Wireless Vuser Recording Options”).

You enter an origin gateway IP address and set the recording mode to CO or CL. Make sure that the recording mode you select is supported by your toolkit or phone.

To record over a phone through wireless connection, you must first dial in to your ISP to get Internet access. Configure the phone to the IP address of the VuGen machine and set the phone to the desired recording mode (CO or CL).

The VuGen machine can exist in one of the following configurations:

- If you connect through a third party ISP, the VuGen machine with the pseudo gateway should be open to Internet access—it must not sit beyond a firewall.
- If you dial in through a Remote Access Server (RAS), you can access the VuGen machine as part of the network.

Bearers Support

The Transport layer protocol in the WAP architecture consists of the Wireless Transaction Protocol (WTP) and the Wireless Datagram Protocol (WDP).

An underlying bearer is a data transport mechanism used to carry the WDP protocols between two devices. Examples of underlying bearers include SMS-CIMD2, UDP, CSD, GSM GPRS, GSM CSD, and Packet Data.

WAP Vusers currently support the UDP (User Datagram Protocol) and SMS-CIMD2 (Short Message Service) bearers.

UDP bearers do not require a separate connection- they operate over an IP network. To work with SMS-CIMD2 however, you must connect to an SMS Center (SMSC) and provide the appropriate information:

- ▶ **IP and Port Information:** For UDP bearers, you define the port and login information in the Run-Time Setting's **Bearers** tab (see "Configuring Bearer Information," on page 755).
- ▶ **Login Information for the SMS Center:** You define the SMS login information in the Run-Time Setting's **Bearers** tab. You can also set this information through the `wap_set_sms_user` function. This is useful for load testing when you need to set the login information for many Vusers using parameterization.
- ▶ **Login Information for the CIMD2:** You set the CIMD2 bearer information in the Run-Time settings **Bearers** tab (see Chapter 61, "Configuring WAP Run-Time Settings").

In some instances, you may need to work with several types of bearers. For example, someone sends you a message in UDP protocol when your phone is off. When you turn your phone on, you retrieve it through the SMS protocol. You can use the `wap_set_bearer` function to switch bearer types during script execution.

RADIUS Support

RADIUS (Remote Authentication Dial-In User Service) is a client/server protocol and software that enables remote access servers to communicate with a central server to authenticate dial-in users and authorize their access to the requested system or service.

RADIUS allows a company to maintain user profiles in a central database that all remote servers can share. It provides better security, allowing a company to set up a policy that can be applied at a single administered network point. Using a central service makes it easier to track usage for billing and store network statistics.

RADIUS has two sub-protocols:

- ▶ **Authentication:** Authorizes and controls user access.
- ▶ **Accounting:** Tracks usage for billing and for keeping network statistics.

In VuGen, the RADIUS protocol is only supported for WSP replay for both Radius sub-protocols—authentication and accounting.

You supply the dial-in information in the Run-Time Settings **Radius** tab. For more information see Chapter 61, “Configuring WAP Run-Time Settings.”

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 simple 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 e-mails, 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.

ProTune 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, ProTune 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 ProTune not to retrieve the message data by configuring the Run-Time settings. For more information, see Chapter 61, “Configuring WAP Run-Time Settings.”

Emulating a Push Initiator

ProTune 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:

- 1** Submit a Push request
- 2** Cancel a Push request
- 3** Submit a query for the status of a push request
- 4** Submit a query for the status of a wireless device’s capabilities
- 5** 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 ProTune. 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 Support

The Multimedia Messaging Service (MMS) is a system application by which a WAP client is able to provide a messaging operation with a variety of media types. MMS Client emulation is implemented with the following Vuser functions.

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

wap_mms_msg_add_field	Adds a field to an MMS message.
wap_mms_msg_add_multipart_entry	Adds a multipart entry to an MMS message.
wap_mms_msg_create	Creates a message for an MMS.
wap_mms_msg_destroy	Destroys an MMS message
wap_mms_msg_submit	Sends a message to the MMS

60

Setting Wireless Vuser Recording Options

Before recording a Wireless session, you can customize the recording options.

This chapter describes:

- ▶ Specifying the Recording Mode (WAP only)
- ▶ Specifying the Information to Record (i-mode and VoiceXML)
- ▶ Specifying a Toolkit

About Setting Recording Options

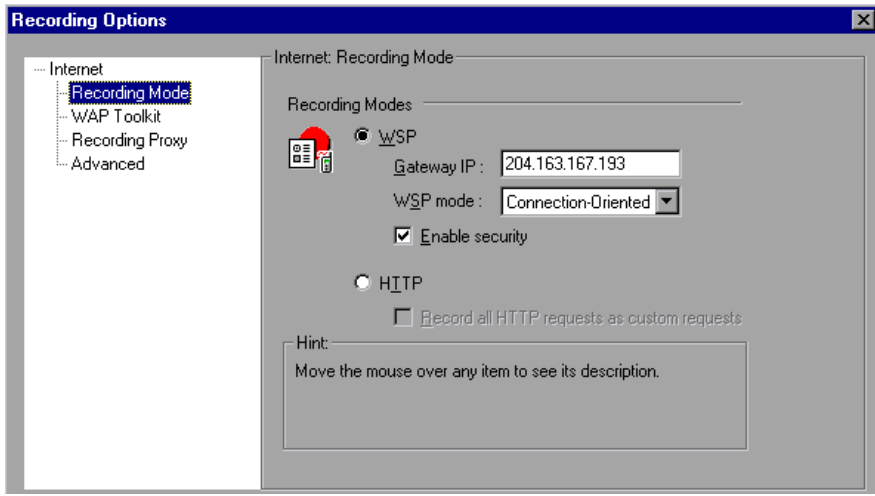
VuGen enables you to generate Wireless Vuser scripts by recording typical processes that users perform on your Web site using their wireless interfaces.

Before recording, you can configure the Recording Options and specify the information to record, the toolkit with which to record, and the global proxy settings.

You can set the common Internet protocol recording options, such as proxy settings and other advanced settings. For more information see Chapter 34, “Setting Recording Options for Internet Protocols.”

Specifying the Recording Mode (WAP only)

Use the **Recording Mode** settings in the Recording Options dialog box (**Tools > Recording Options**) to define the information that VuGen records during a recording session for WAP Vusers.



To define recording information for WAP Vusers:

Select one of the following options in the **Recording Mode** section:

- **WSP:** Instructs VuGen to record all WSP traffic between the toolkit or phone and the gateway. The actions are recorded as URL steps. Enter the IP address of the gateway, and select **Connectionless** or **Connection-Oriented** from the WSP mode box. To allow recording using secure WAP, select the **Enable security** check box.

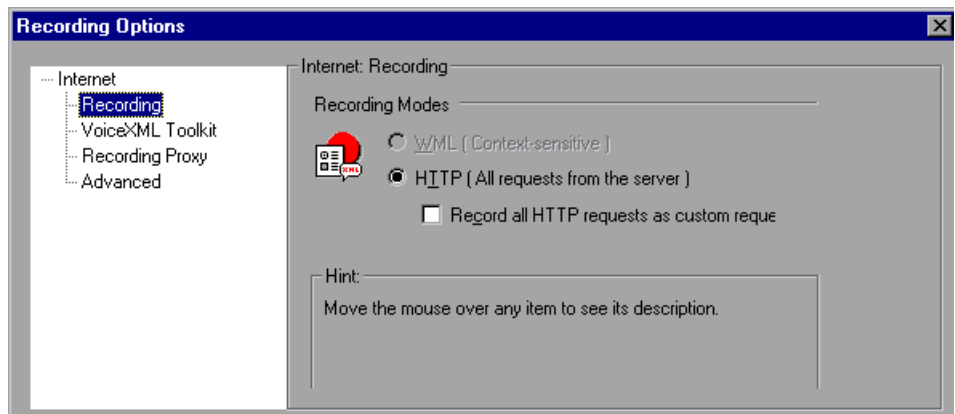
For recording in WSP mode, VuGen contains native support for the *Phone.com UP Simulator 4.1* toolkit. It detects the installation, automatically sets the configuration parameters, and launches it. For Nokia toolkits (1.3 and 2.0), you need to launch the toolkit manually assigning it the proper IP address. For more information, see “Troubleshooting Wireless Vuser Scripts,” on page 738.

- **HTTP:** Instructs VuGen to record the HTTP traffic between the toolkit and the Web server as URL steps. Select the **Record all HTTP requests as custom requests** check box to record all HTTP requests as contextless custom HTTP requests, generating `web_custom_request` functions.

Specifying the Information to Record (i-mode and VoiceXML)

Use the **Recording** node in the Recording Options tree to define the information that VuGen records during a recording session for i-mode or VoiceXML Vusers.

The only available recording mode is **HTTP**. This mode instructs VuGen to record the HTTP traffic between the toolkit and the Web server as URL steps.

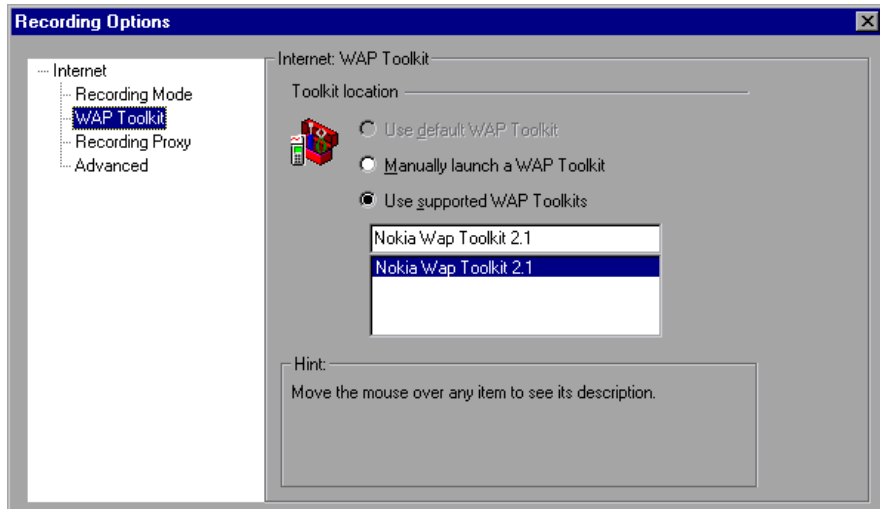


To define recording information for i-mode Vusers:

- 1 Open the Recording Options (**Tools > Recording Options**) and in the Recording Options tree, select the **Internet Protocol:Recording** node.
- 2 In the **Information to record** section, select the **Record all HTTP requests as custom requests** option to record all HTTP requests as contextless Custom HTTP Requests, generating `web_custom_request` functions.

Specifying a Toolkit

You can specify which toolkit VuGen should use when recording a Wireless Vuser script. You use the **Toolkit** node in the Recording Options tree to specify the desired WAP, i-mode, or VoiceXML toolkit.



To specify the toolkit for recording a Wireless Vuser script:

- 1 Choose **Tools > Recording Options** and select the **WAP Toolkit (or i-mode VoiceXML Toolkit)** node.
- 2 Select one of the following options in the **Toolkit Location** section:
 - **Use default WAP (i-mode or VoiceXML) Toolkit:** Instructs VuGen to use the default toolkit on the recording computer (currently disabled).
 - **Manually launch a WAP (i-mode or VoiceXML) Toolkit:** Instructs VuGen not to launch a toolkit when you start recording. You must manually launch a WAP toolkit after you start the recording session.
 - **Use supported WAP (i-mode or VoiceXML) Toolkits:** Instructs VuGen to use a specific toolkit installed on the machine. Select one of the available toolkits listed in the dialog box.

61

Configuring WAP Run-Time Settings

After you record a WAP Vuser script, you configure the WAP specific run-time settings.

This chapter describes:

- ▶ Configuring Gateway Options
- ▶ Configuring Bearer Information
- ▶ Configuring RADIUS Connection Data

For information on setting the common Internet protocol run-time settings for WAP and all Wireless protocols, see Chapter 36, “Configuring Internet Run-Time Settings.”

About WAP Run-Time Settings

After developing a WAP Vuser script, you set the WAP specific run-time settings. These settings let you control the behavior of the WAP Vusers so that they accurately emulate real users on a WAP device. You can configure WAP run-time settings in the areas of Gateway, Radius, and Bearer settings.

You set the WAP run-time settings from the Run-Time Settings dialog box. You click on the appropriate tab to view and specify the desired settings.

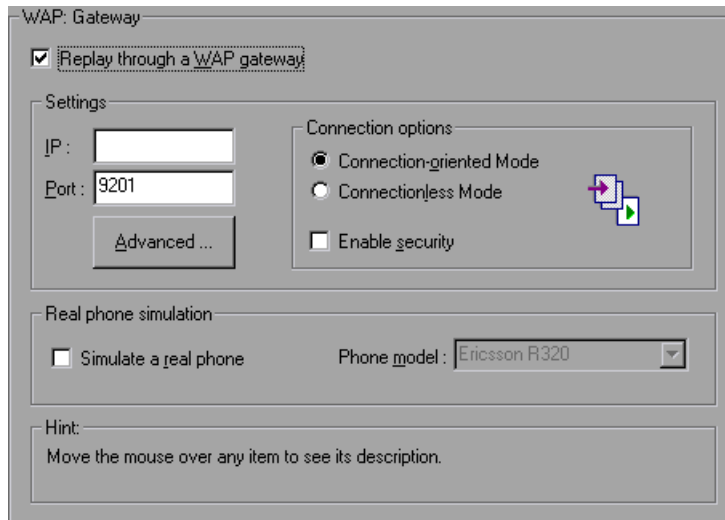
To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar. You can also modify the run-time settings from the ProTune Console.



This chapter discusses the Gateway Run-Time settings for WAP Vusers. For information about the general run-time settings that apply to all wireless Vusers, see Chapter 36, “Configuring Internet Run-Time Settings.”

Configuring Gateway Options

You use the **WAP:Gateway** node in the Run-Time Settings tree to set the gateway settings:



The Gateway settings are only relevant if you want to run the Vusers using WSP protocol, accessing a Web server via a WAP Gateway (by selecting the **Replay through a WAP gateway** option). If you are running a script through a gateway, you must specify an IP and port address.

If you are running Vusers in the HTTP mode, accessing a Web server directly, (by clearing the **Replay through a WAP gateway** option), then the Gateway settings do not apply.

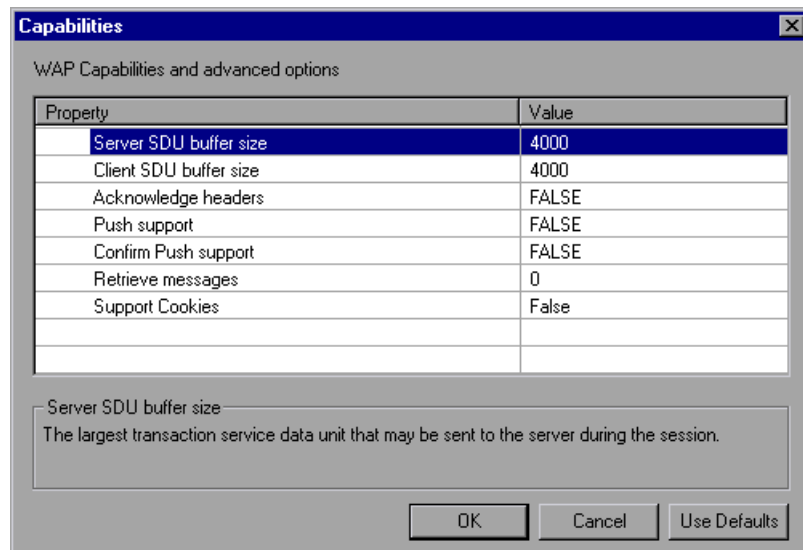
Setting a Gateway Address and Connection Options

When running your Vusers through a WAP gateway, VuGen automatically sets default port numbers, depending on the selected mode. However, you can customize the settings and specify a custom IP address and port for the gateway.

You can also indicate the desired replay connection mode. The supported connections modes are Connection-Oriented (CO) or Connectionless mode (CL). In addition, you can indicate whether or not the connection should be secure.

Setting Advanced Gateway Settings

In the Capabilities dialog box, you can configure the WAP Capabilities and advanced gateway options:



- **Server SDU size** - the largest transaction service data unit that may be sent to the *server* during the session (4000 by default).
- **Client SDU size** - the largest transaction service data unit that may be sent to the *client* during the session (4000 by default).
- **Acknowledgement Headers** - return standard headers that provide information to the gateway (disabled by default).

- ▶ **Push Support** - Enables push type messages across the gateway (disabled by default).
- ▶ **Confirm Push Support** - In CO mode, if a push message is received, this option instructs the Vuser to confirm the receipt of the message (disabled by default).
- ▶ **Retrieve Messages** - When a push messages is received, this option instructs the Vuser to retrieve the message data from the URL indicated in the push message (disabled by default).
- ▶ **Support Cookies** - Provide support for saving and retrieving cookies (disabled by default).

Real Phone Simulation

VuGen allows you to indicate the type of phone instrument for replaying the Vusers. You can choose from a list of phones from popular vendors. VuGen determines the correct client headers for the selected phone and emulates it accordingly. Note that when you enable real phone simulation, all of the Advanced gateway options are ignored. Instead it retrieves the header and client capability information from the Vugen configuration file which defines each of the supported phones.

Real Phone simulation is especially useful if you need to perform a tests for several different telephones. For example, you can record a script for a *Motorola Timeport*, and replay it on a *Nokia 6110*. When you replay a script using Real Phone simulation, it ignores all of the **wap_set_capability** and **wap_add_const_header** functions in the script. It retrieves all the necessary information from the configuration file which defines the headers for each phone.

If the phone you want to emulate does not appear on the list, you can add it to the Run-Time settings interface by manually adding it to the configuration file, *LrWapPhoneDB.dat* in VuGen's *dat* directory. For more information, see the comments at the beginning of the configuration file.

To set the WAP gateway options:

- 1** Click the **Run-Time Settings** button or select **Vuser > Run-Time Settings** to display the Run-Time Settings dialog box. Select the **WAP:Gateway** node.
- 2** To replay the script in WSP mode (not HTTP), select **Replay through a WAP gateway**.
- 3** Specify an IP address and port for the gateway. You can also use the default port indicated by VuGen.
- 4** Select a connection mode—**Connection-Oriented** or **Connectionless**. To indicate a secure connection mode, select the **Use secure connection** option.
- 5** To emulate a popular phone, select **Simulate a Real Phone** and select the desired phone from the pull-down list.
- 6** If you are not emulating a popular phone, click **Advanced** to set the client capabilities and other advanced gateway options.
 - Enter values for the **Server SDU** and **Client SDU**.
 - To instruct Vusers to retrieve Acknowledgement Headers, select the **Acknowledgement Headers** option.
 - To allow push messages, choose **True** in the column adjacent to **Push Support**.
 - To allow confirmations for push messages, select **True** in the column adjacent to **Confirm Push Support**.
 - To retrieve data from push message URLs, select **True** in the column adjacent to **Retrieve Messages**.
 - To enable cookies, select **True** in the column adjacent to **Support Cookies**.

Configuring Bearer Information

ProTune supports both UDP and SMS bearers. In the Run-Time settings you indicate the initial bearer. You can switch between bearers during replay using the `wap_set_bearer` function. Note that if you want to use both bearers, you must enable them before replay in the Run-Time settings.

To work with the SMS-CIMD2 bearer, you must connect to a Short Message System Center (SMSC) and provide login information. You define the port information in the Run-Time Settings **WAP:Bearerers** node.

You can set the SMS login information using the **wap_set_sms_user** API function or through the run-time settings. The advantage of setting the login information through a function, is that you can use parameters to run the script. with many values. The values in the API function override the run-time settings. You set the bearer attributes in the Run-Time Settings **Bearerers** node:

Bearer Setting	Description
Enable UDP Bearer	Open a connection to a UDP bearer
Enable SMS-CIMD2 Bearer	Open a connection to a CIMD2 bearer
Bearer Type	The default bearer type (UDP or CIMD2).
CIMD2: IP Address	IP address of the SMSC server.
CIMD2: Port Number	Port number of the SMSC server.
CIMD2: Gateway ID	WAP gateway application ID as defined in the SMSC.
CIMD2: User Name	The username for logging on to the server.
CIMD2: User Password	The user's password.
CIMD2: Originating Address	User originating address.

To set the WAP Bearer options:



- 1 Click the **Run-Time Settings** button or select **Vuser > Run-Time Settings** to display the Run-Time Settings dialog box. Select the **Bearer** node.

WAP: Bearers

Settings

Property	Value
Enable UDP bearer	True
Enable SMS-CIMD2 bearer	False
Bearer type	UDP
CIMD2: IP address	0.0.0.0
CIMD2: Port number	1
CIMD2: Gateway ID	1
CIMD2: User name	username
CIMD2: User password	password
CIMD2: Originating Address	1

Enable UDP bearer
Allow the Vuser to use UDP as the underlying bearer.

- 2 Select True to enable the desired bearers: **UDP** or **SMS-CIMD2**.
- 3 Select a **Bearer type** that you want to begin with, from the right column—UDP or SMS-CIMD2.
- 4 Enter the **SMSC IP address** in the dot form.
- 5 Enter the **SMSC Port number**.
- 6 Enter the **SMSC Gateway ID**—not the SMS Gateway ID.
- 7 Enter the **SMSC User name**.
- 8 Enter the **SMSC User password**.
- 9 Enter the **SMSC originating address**.
- 10 Click **OK** to accept the setting and close the dialog box.

Configuring RADIUS Connection Data

RADIUS (Remote Authentication Dial-In User Service) is a client/server protocol and software that enables remote access servers to communicate with a central server to authenticate dial-in users and authorize their access to the requested system or service.

You supply the dial-in information in Run-Time Settings' **Radius** node:

Radius Authentication	A flag indicating whether or not to use the authentication protocol.
IP Address	IP address of the Radius server.
Authentication Port Number	Authentication port of the Radius server.
Accounting Port Number	Accounting port of the Radius server.
Secret Key	The secret key of the Radius server.
User Name	The username for logging on to the Radius server.
User Password	The user's password.
MSISDN	MSISDN number.
Network Type	Accounting network type: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).

To set the WAP Radius options:



- 1 Click the **Run-Time Settings** button or select **Vuser > Run-Time Settings** to display the Run-Time Settings dialog box. Click the **Radius** node.

WAP: Radius

Settings

Property	Value
Network type	CSD
IP address	0.0.0.0
Authentication port number	1812
Accounting port number	1813
Secret key	secret

Network type

Select one of the network types

- 2 In the Radius Authentication section, choose **True** to enable authentication, or **False** to disable it.
- 3 Enter the **IP address** of the Radius server in dot form.
- 4 Enter the **Authentication Port number** and **Accounting Port number** of the Radius server.
- 5 Type in the **Secret key** for Radius or Accounting Authentication.
- 6 Enter the **User name** for logging into the Radius server.
- 7 Enter the **User password** for the user name.
- 8 Enter the **MSISDN** identification number.
- 9 Choose an accounting **Network type**: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).
- 10 Click **OK** to accept the settings and close the dialog box.

Part XVI

Information for Advanced Users

62

Creating Vuser Scripts in Visual Studio

You can create a Vuser script template in Visual Studio using Visual C or Visual Basic. You compile it as you would a regular C or Visual Basic program.

This chapter describes:

- ▶ Creating a Vuser Script with Visual C
- ▶ Creating a Vuser Script with Visual Basic
- ▶ Configuring Runtime Settings and Parameters

About Creating Vuser Scripts in Visual Studio

There are several ways to create Vuser scripts: through VuGen or a development environment such as Visual Studio.

<i>VuGen</i>	You can use VuGen to create Vuser script that run on Windows platforms by recording or by manually programming within the VuGen editor.
<i>Visual Studio</i>	For users working with Visual Studio, you can program in Visual Basic, C or C++. The programs must be compiled into a dynamic link library (dll).

This appendix describes how to develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the ProTune libraries.

You can also program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes. Programming within VuGen is available for C, Java, Visual Basic, VBScript, and JavaScript. For more information, see Chapter 22, "Creating Custom Vuser Scripts."

To create a Vuser script through programming, you can use a VuGen template as a basis for a larger Vuser script. The template provides:

- correct program structure
- VuGen API calls
- source code and makefiles for creating a dynamic library

After creating a basic Vuser script from a template, you can enhance the script to provide run-time information and statistics. For more information, see Chapter 6, "Enhancing Vuser Scripts."

An online C reference of the common functions used in Vuser scripts, are included in the *Online Function Reference* (**Help > Function Reference**)

Creating a Vuser Script with Visual C

Please note that you can create Vuser scripts using Visual C version 6.0 or higher.

To create a Vuser script with Visual C:

- 1** In Visual C, create a new project - dynamic link library (dll). Choose **File > New** and click the Projects tab.
- 2** In the Wizard, choose *empty dll*.
- 3** Add the following files to the project:
 - A new *cpp* file with 3 exported function: *init*, *run*, *end* (the names may be customized).
 - A library *:lrun50.lib* (located in the <lr installation dir>/lib).
- 4** In the project settings change the following:
 - Select the C/C++ tab and choose **Code generation** (Category) > **Use Run Time library** (List). Change it to: **Multithreaded dll**.

- Select the C/C++ tab and choose **Preprocessor** (Category) > **Preprocessor definitions** (edit field) Remove `_DEBUG`.
- 5** Add code from your client application, or program as you normally would.
- 6** Enhance your script with General VuGen functions. For example, **lr_output_message** to issue messages, **lr_start_transaction** to mark transactions, and so forth. For more information, see the General functions in the *Online Function Reference* (**Help** > **Function Reference**).
- 7** Build the project. The output will be a DLL.
- 8** Create a directory with the same name as the DLL and copy the DLL to this directory.
- 9** In the **lrvuser.usr** file in the *Template* directory, Update the USR file key *BinVuser* with the DLL name: `BinVuser=<DLL_name>`.

In the following example, the **lr_output_message** function issues messages indicating which section is being executed. The **lr_eval_string** function

retrieves the name of the user. To use the following sample, verify that the path to the VuGen include file, *lrun.h* is correct.

```
#include "c:\mercury\lrun_5\include\lrun.h"

extern "C" {
int __declspec(dllexport) Init (void *p)
{
    lr_output_message("in init");
return 0;
}

int __declspec(dllexport) Run (void *p)
{
    const char *str = lr_eval_string("<name>");
    lr_output_message("in run and parameter is %s", str);
return 0;
}

int __declspec(dllexport) End (void *p)
{
    lr_output_message("in end");
return 0;
}
} //extern C end
```

Creating a Vuser Script with Visual Basic

To create a Vuser in Visual Basic:

- 1** In Microsoft Visual Basic, create a new project. Select **File > New Project**.
- 2** Select **Virtual User**. A new project is created with one class and a template for a Vuser.
- 3** Save the project before you continue to program. Chose **File > Save Project**.

- 4 Open the Object Browser (**View** menu). Select the *Vuser* library and double-click on the Vuser Class module to open the template. The template contains three sections, *Vuser_Init*, *Vuser_Run*, and *Vuser_End*.

```
Option Explicit

Implements Vuser

Private Sub Vuser_Init()
'Implement the Vuser initialization code here
End Sub

Private Sub Vuser_Run()
'Implement the Vuser main Action code here
End Sub

Private Sub Vuser_End()
'Implement the Vuser termination code here
End Sub
```

- 5 Add code from your client application, or program as you normally would.
- 6 Use the Object Browser to add the desired VuGen elements to your code, such as transactions, think time, rendezvous, and messages, using the object browser.
- 7 Enhance your program with run-time settings and parameters. For more information, see “Configuring Runtime Settings and Parameters” on page 767.
- 8 Build the Vuser script: select **File > Make** *project_name.dll*.

The project is saved in the form of a Vuser script (.usr). The script resides in the same directory as the project.

Configuring Runtime Settings and Parameters

After you create the DLL for your script, you create a script (.usr) and configure its settings. The *lrbin.bat* utility provided with VuGen lets you define parameters and configure runtime settings for scripts created with

Visual C and Basic. This utility is located in the *bin* directory of the VuGen installation.

To configure runtime settings and parameterize scripts:

- 1 In the VuGen *bin* directory, double-click on *lrbin.bat*. The Standalone Vuser Configuration dialog box opens.



- 2 Choose **File > New**. Specify a script name for the *usr* file. The script name must be identical to the name of the directory to which you saved the DLL.
- 3 Choose **Vuser > Advanced** and enter the DLL name in the Configuration dialog box.
- 4 Choose **Vuser > Run-time Settings** to define run-time settings. The Run-time Settings dialog box is identical to that displayed in the VuGen interface. For more information, see Chapter 9, "Configuring Run-Time Settings."
- 5 Choose **Vuser > Parameter List** to define parameters for your script. The Parameter dialog boxes are identical to those in VuGen. For more information, see Chapter 7, "Defining Parameters."

Test the script by running it in standalone mode. Choose **Vuser > Run Vuser**. The Vuser execution window appears while the script runs.

- 6 Choose **File > Exit** to close the configuration utility.

63

Programming with the XML API

You can create Vuser scripts that support the complete XML structure. VuGen provides functions that allow you to query and manipulate the XML data.

This chapter describes:

- ▶ Understanding XML Documents
- ▶ Using XML Functions
- ▶ Specifying XML Function Parameters
- ▶ Working with XML Attributes
- ▶ Structuring an XML Script
- ▶ Enhancing a Recorded Session

The following information applies primarily to Web, SOAP, and Wireless Vuser scripts.

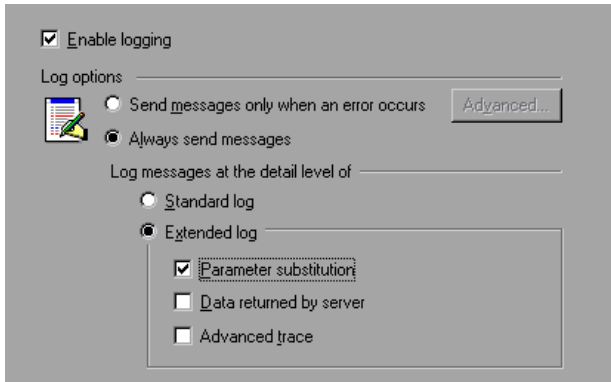
About Programming with the XML API

VuGen's support for XML allows you to dynamically work with XML code and retrieve the values during test execution. Follow these steps in creating an effective XML script:

- ▶ Record a script in the desired protocol, usually Web, SOAP, or Wireless.
- ▶ Copy the XML structures into your script.
- ▶ Add XML functions from the LR API in order to retrieve dynamic data and the XML element values.

The LR API uses XPath, the XML Path language to manipulate the text in an XML document.

You can instruct VuGen to display the output values of XML elements in the Execution log window using the Run-Time settings. VuGen displays the line numbers, the number of matches, and the value. To allow the displaying of values, you need to enable parameter substitution. In the Run-Time settings, open the **General:Log** node, select **Extended log**, and choose **Parameter Substitution**. For more information, see Chapter 9, “Configuring Run-Time Settings.”



All ProTune XML functions return the number of matches successfully found, or zero for failure.

Understanding XML Documents

XML, or Extensible Markup Language, is a markup language that you can use to create your own custom tags. Using these tags, you give a meaning to the text between the tags. This stands in contrast to standard HTML tags such as H1, P, DIV, etc. which cannot be customized and do not indicate the content of the text.

XML documents consist of trees with many nodes and branches. There are three common terms used that describe the parts of an XML document: *tags*, *elements*, and *attributes*. The following example illustrates these terms:

```

<acme_org>
  <accounts_dept>
    <employee type='PT'>
      <name>John Smith</name>
      <cubicle>227</cubicle>
      <extension>2145</extension>
    </employee>
  </accounts_dept>
  <engineering_dept>
    <employee type='PT'>
      <name>Sue Jones</name>
      <extension>2375</extension>
    </employee>
  </engineering_dept>
</acme_org>

```

A *tag* is the text between the left and right angle brackets. `<acme_org>`, `<employee>` and `<name>` are examples of tags. There are starting tags, such as `<name>`, and ending tags, such as `</name>`. The above XML fragment describes the Acme organization with two employees, John Smith and Sue Jones.

An *element* is the starting tag, ending tag, and everything in between. In the sample above, the `<employee>` element contains three child elements: `<name>`, `<cubicle>`, and `<extension>`.

An *attribute* is a name-value pair inside the starting tag of an element. In this example, `type='PT'` is an attribute of the `<employee>` element;

In the above example, the tag *name* is an element of *employee*. Each element has a value. An example of a *name* element's value is the string "John Smith".

Using XML Functions

The next sections provide examples of how to work with data in an XML tree. Certain functions allow you to retrieve information, and others let you write information to an XML tree. These examples use the following XML tree containing the names and extensions of several employees in the *Acme* organization.

```
<acme_org>
  <accounting_dept>
    <employee type='PT'>
      <name>John Smith</name>
      <extension>2145</extension>
    </employee>
  </accounting_dept>
  <engineering_dept>
    <employee type='PT'>
      <name>Sue Jones</name>
      <extension>2375</extension>
    </employee>
  </engineering_dept>
</acme_org>
```

Reading Information from an XML Tree

The functions which read information from an XML tree are:

lr_xml_extract	Extracts XML string fragments from an XML string
lr_xml_find	Performs a query on an XML string.
lr_xml_get_values	Retrieves values of XML elements found by a query

To retrieve a specific value through a query, you specify the tags of the parent and child nodes in a path format.

For example, to retrieve an employee name in the Accounting department, use the following string:

```
lr_xml_get_values("XML={XML_Input_Param}",
  "ValueParam=OutputParam",
  "Query=/acme_org/accounting_dept/employee/name",
  LAST);
```

The Execution log window (with Extended logging enabled) shows the output of this function:

Output:

Action.c(20): "lr_xml_get_values" was successful, 1 match processed

Action.c(25): Query result = **John Smith**

Writing to an XML Structure

The functions which write values to an XML tree are:

lr_xml_delete	Deletes fragments from an XML string
lr_xml_insert	Inserts a new XML fragment into an XML string
lr_xml_replace	Replaces fragments of an XML string
lr_xml_set_values	Sets the values of XML elements found by a query
lr_xml_transform	Applies Extensible Stylesheet Language (XSL) transformation to XML data.

The most common *writing* function is **lr_xml_set_values** which sets the values of specified elements in an XML string. The following example uses **lr_xml_set_values** to change the phone extensions of two *employee* elements in an XML string.

First, we save the XML string to a parameter called *XML_Input_Param*. We want two values to be matched and substituted, so we prepare two new parameters, *ExtensionParam_1* and *ExtensionParam_2*, and set their values to two new phone extensions, 1111 and 2222.

lr_xml_set_values contains the argument “ValueName=ExtensionParam”, which picks up the values of *ExtensionParam_1* and *ExtensionParam_2*. The current extensions of the two employees are substituted with the values of these parameters, 1111 and 2222. The value of *OutputParam* is then evaluated proving that the new phone extensions were in fact substituted.

```
Action() {

    int i, NumOfValues;
    char buf[64];

    lr_save_string(xml_input, "XML_Input_Param"); // Save input as parameter
    lr_save_string("1111", "ExtensionParam_1");
    lr_save_string("2222", "ExtensionParam_2");

    lr_xml_set_values("XML={XML_Input_Param}",
        "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
        "SelectAll=yes", "Query=//extension", LAST);

    NumOfValues= lr_xml_get_values("XML={NewXmlParam}",
        "ValueParam=OutputParam", "Query=//extension",
        "SelectAll=yes", LAST);

    for (i = 0; i < NumOfValues; i++) { /* Print the multiple values of MultiParam */

        sprintf(buf, "Retrieved value %d : {OutputParam_%d}", i+1, i+1);
        lr_output_message(lr_eval_string(buf));
    }

    return 0;
}
```

Output:

Action.c(40): Retrieved value 1: 1111

Action.c(40): Retrieved value 2: 2222

Specifying XML Function Parameters

Most XML API functions require that you specify the **XML element** and a **query**. You can also indicate if you want to retrieve all results or a single one.

Defining the XML Element

For defining the XML element to query, you can specify a literal string of the XML element, or a parameter that contains the XML. The following example shows the XML input string defined as a literal string:

```
"XML=<employee>JohnSmith</employee>
```

Alternatively, the XML string can be a parameter containing the XML data. For example:

```
"XML={EmployeeNameParam}"
```

Querying an XML Tree

Suppose you want to find a value within an XML tag, for example, an employee's extension. You formulate a query for the desired value. The query indicates the location of the element and which element you want to retrieve or set. The path that you specify limits the scope of the search to a specific tag. You can also search for all elements of a specific type under all nodes below the root.

For a specific path, use `"Query=/full_xml_path_name/element_name"`

For the same element name under all nodes, use `"Query=//element_name"`

In the ProTune implementation of XML functions, the scope of a query is the entire XML tree. The tree information is sent to the ProTune API functions as the value of the *xml* argument.

Multiple Query Matching

When you perform a query on an XML element, by default VuGen returns only the first match. To retrieve multiple values from a query, you specify the `"SelectAll=yes"` attribute within your functions. VuGen adds a suffix of `_index` to indicate multiple parameters. For example, if you defined a

parameter by the name *EmployeeName*, VuGen creates *EmployeeName_1*, *EmployeeName_2*, *EmployeeName_3*, etc.

```
lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);
```

With functions that *write* to a parameter, the values written to the parameter can then be evaluated. For example, the following code retrieves and prints multiple matches of a query:

```
NumOfValues = lr_xml_get_values("Xml={XmlParam}", "Query=//name",
"SelectAll=yes", "ValueParam=EmployeeName", LAST);
```

For functions that *read* from parameters, the values of the parameters must be pre-defined. The parameter must also use the convention *ParamName_IndexNumber*, for example *Param_1*, *Param_2*, *Param_3*, etc. This collection of parameters is also known as a parameter set.

In the following example, **lr_xml_set_values** reads values from the parameter set and then uses those values in the XPath query. The parameter set that represents the employee extensions, is called *ExtensionParam*. It has two members: *ExtensionParam_1* and *ExtensionParam_2*. The **lr_xml_set_values** function queries the XML input string and sets the value of the first match to 1111 and the second match to 2222.

```
lr_save_string("1111", "ExtensionParam_1");
lr_save_string("2222", "ExtensionParam_2");
```

```
lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);
```


Working with XML Attributes

VuGen contains support for attributes. You can use a simple expression to manipulate attributes of XML elements and nodes, just as you can manipulate the elements themselves.

You use this expression to manipulate the desired attribute or only attributes with specific values. In the following example, `lr_xml_delete` deletes the first *cubicle* element with the *name* attribute.

```
lr_xml_delete("Xml={ParamXml}",
              "Query=//cubicle/@name",
              "ResultParam=Result",
              LAST
            );
```

In the next example, `lr_xml_delete` deletes the first cubicle element with a *name* attribute that is equal to *Paul*.

```
lr_xml_delete("Xml={ParamXml}",
              "Query=//cubicle/@name="Paul",
              "ResultParam=Result",
              LAST
            );
```

Structuring an XML Script

Initially, you create a new script in the desired protocol (HTTP, SOAP, etc.). You can record a session in that protocol, or you may program the entire script without recording. Structure the Actions section of the script as follows:

- XML input declaration
- The Actions section

The **XML input section** contains the XML tree that you want to use as an input variable. You define the XML tree as a **char** type variable. For example:

```

char *xml_input=
"<acme_org>"
  "<employee>"
    " <name>John Smith</name>"
    "<cubicle>227</cubicle>"
    "<extension>2145</extension>"
  "</employee>"
  "<employee>"
    "<name>Sue Jones</name>"
    "<cubicle>227</cubicle>"
    "<extension>2375</extension>"
  "</employee>"
"</acme_org>";

```

The **Action section** contains the evaluation of the variables and queries for the element values. In the following example, the xml input string is evaluated using **lr_save_string**. The input variable is queried for employee names and extensions.

```

Action() {

    /* Save the input as a parameter.*/
    lr_save_string(xml_input, "XML_Input_Param");

    /* Query 1 - Retrieve an employee name from the specified element.*/
    lr_xml_get_values("XML={XML_Input_Param}",
        "ValueParam=OutputParam",
        "Query=/acme_org/employee/name", LAST);

    /* Query 2 - Retrieve an extension under any path below the root.*/
    lr_xml_get_values("XML={XML_Input_Param}",
        "ValueParam=OutputParam",
        "Query=//extension", LAST);

    return 0;
}

```

Enhancing a Recorded Session

You can prepare an XML script by recording a session and then manually adding the relevant XML and ProTune API functions.

The following SOAP protocol example illustrates how a recorded session was enhanced with ProTune API functions. Note that the only function that was recorded was **web_submit_data**, which appears in bold.

The first section contains the XML input declaration of the variable SoapTemplate, for a SOAP message.

```
#include "as_web.h"

// SOAP message
const char*pSoapTemplate=
    "<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
    "<soap:Body>"
        "<SendMail xmlns=\"urn:EmailPortTypeInft-IEmailService\"/>"
    "</soap:Body>"
"</soap:Envelope>";
```

The following section represents the actions of the user.

```
Action1()
{
    // get response body
    web_reg_save_param("ParamXml", "LB=", "RB=", "Search=body", LAST);

    // fetch weather by HTTP GET
    web_submit_data("GetWeather",
        "Action=http://glkev.net.innerhost.com/glkev_ws/
        WeatherFetcher.aspx/GetWeather",
        "Method=GET",
        "EncType=",
        "RecContentType=text/xml",
        "Referer=http://glkev.net.innerhost.com
        /glkev_ws/WeatherFetcher.aspx?op=GetWeathe
r",
```

```

    "Snapshot=t2.inf",
    "Mode=HTTP",
    ITEMDATA,
    "Name=zipCode", "Value=10010", ENDITEM,
    LAST);

// Get City value
lr_xml_get_values("Xml={ParamXml}",
    "Query=City",
    "ValueParam=ParamCity",
    LAST
);

lr_output_message(lr_eval_string("***** City = {ParamCity} *****"));

// Get State value
lr_xml_get_values("Xml={ParamXml}",
    "Query=State",
    "ValueParam=ParamState",
    LAST
);

lr_output_message(lr_eval_string("***** State = {ParamState} *****"));

// Get several values at once by using template
lr_xml_get_values_ex("Xml={ParamXml}",
    "Template="
        "<Weather>"
        "    <Time>{ParamTime}</Time>"
        "    <Temperature>{ParamTemp}</Temperature>"
        "    <Humidity>{ParamHumid}</Humidity>"
        "    <Conditions>{ParamCond}</Conditions>"
        "</Weather>",
    LAST
);

lr_output_message(lr_eval_string("***** Time = {ParamTime}, Temperature =
                                {ParamTemp}, "

```

```
"Humidity = {ParamHumid}, Conditions =
  {ParamCond} *****");
```

```
// Generate readable forecast
```

```
lr_save_string(lr_eval_string("\r\n\r\n*** Weather Forecast for {ParamCity}, {ParamState} ***\r\n"
    "\tTime: {ParamTime}\r\n"
    "\tTemperature: {ParamTemp} deg. Fahrenheit\r\n"
    "\tHumidity: {ParamHumid}\r\n"
    "\t{ParamCond} conditions expected\r\n"
    "\r\n"),
    "ParamForecast"
);
```

```
// Save soap template into parameter
```

```
lr_save_string(pSoapTemplate, "ParamSoap");
```

```
// Insert request body into SOAP template
```

```
lr_xml_insert("Xml={ParamSoap}",
    "ResultParam=ParamRequest",
    "Query=Body/SendMail",
    "position=child",
    "XmlFragment="
        "<FromAddress>taurus@merc-int.com</FromAddress>"
        "<ToAddress>support@merc-int.com</ToAddress>"
        "<ASubject>Weather Forecast</ASubject>"
        "<MsgBody/>",
    LAST
);
```

```
//
//   "<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">"
//   "<soap:Body>"
//   "<SendMail xmlns="urn:EmailPortTypeInft-IEmailService"/>"
//   "<FromAddress>taurus@merc-int.com</FromAddress>"
//   "<ToAddress>support@merc-int.com</ToAddress>"
//   "<ASubject>Weather Forecast</ASubject>"
//   "<MsgBody/>"
//   "</SendMail>"
```

```

//      "</soap:Body>"
//      "</soap:Envelope>";
//

// Insert actual forecast text
lr_xml_set_values("Xml={ParamRequest}",
                 "ResultParam=ParamRequest",
                 "Query=Body/SendMail/MsgBody",
                 "ValueParam=ParamForecast",
                 LAST);

// Add header for SOAP
web_add_header("SOAPAction", "urn:EmailPortTypeInft-IEmailService");

// Get response body
web_reg_save_param("ParamXml", "LB=", "RB=", "Search=body", LAST);

// Send forecast to recipient, using SOAP request
web_custom_request("web_custom_request",
                  "URL=http://webservices.matlus.com/scripts/emailwebservice.dll/soap/IEmailservice",
                  "Method=POST",
                  "TargetFrame=",
                  "Resource=0",
                  "Referer=",
                  "Body={ParamRequest}",
                  LAST);

// Verify that mail was sent
lr_xml_find("Xml={ParamXml}",
           "Query=Body/SendMailResponse/return",
           "Value=0",
           LAST
);

return 0;
}

```

64

VuGen Debugging Tips

This chapter contains a few methods for obtaining more detailed debugging information to help you produce error-free Vuser scripts.

- General Debugging Tip
- Using C Functions for Tracing
- Examining Replay Output
- Debugging Database Applications
- Working with Oracle Applications
- Solving Common Problems with Oracle Applications
- Two-tier Database Scripting Tips

General Debugging Tip

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

Using C Functions for Tracing

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

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

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

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

To view the output of the trace information, use the print callback function, `ci_set_print_CB()`.

Adding Additional C Language Keywords

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

To add additional keywords:

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

The file has the following format:

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


Examining Replay Output

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

Debugging Database Applications

The following tips apply to database applications only (Oracle, ODBC, Ctlib):

- Generating Debugging Information
- Examining Compiler Information

Generating Debugging Information

Note: You can now set options to view most of the information described in this section using VuGen's user interface.

VuGen contains an inspector "engine." You can force VuGen recorder to create "inspector" output by editing `\WINDOWS_DIR\vugen.ini` as follows:

```
[LogMode]
EnableAscii=ASCII_LOG_ON
```

When this option is enabled, VuGen creates a file, **vuser.asc** in the Data directory at the end of the recording. Note that this option should be used for debugging purposes only. This output file can become very large (several MB) and have serious effects on machine performance and disk space.

For cases like ODBC-based applications, it is possible to configure the ODBC Administrator (located in the Windows Control Panel) to provide a similar trace output. Open the ODBC options, and select 'Trace ODBC calls' to ON. Similarly the ODBC Developer Kit provides a Spy utility for call tracing.

To enable further debug information, add the following section to the `\WINDOWS_DIR\vugen.ini` file

```
[INSPECTOR]
TRACE_LEVEL=3
TRACE_FILENAME=c:\tmp\sqltrace.txt
```

The file (`sqltrace.txt`) will include useful internal information regarding the hooking calls made during recording. The `trace_level` is between 1 and 3, with 3 representing the most detailed debug level. Note that in VuGen versions 5.02 and higher, you can set the trace level from the user interface.

Examining Compiler Information

You can view information about each stage of code generation, preprocessing and compilation to determine the source of any errors.

Code Generation Information

Look at the `vuser.log` file under the Data directory. This file, which contains a log of the code generation phase, is automatically created at the end of every lrd recording (i.e. all database protocols). The following is an example of a log file:

```
lrd_init: OK
lrd_option: OK
lrd_option: OK
lrd_option: OK
Code generation successful
lrd_option: OK
lrd_end: OK
```

If any of the messages are not OK or successful, then a problem occurred during the code generation.

Preprocessing and Compilation Information

During runtime, VuGen displays information about both the preprocessing and compilation processes.

Working with Oracle Applications

Oracle Applications is a two-tier ("fat" client) packaged application, made up of 35 different modules (Oracle Human Resources, Oracle Financials, and so forth).

There are a number of issues that you should be aware of while recording and replaying users for Oracle Applications:

- ▶ A typical script contains thousands of events, binds and assigns.
- ▶ A typical script has many db connections per user session.
- ▶ scripts almost always require correlated queries.
- ▶ Oracle Applications' clients are 16-bit only (developed with Oracle Developer 2000). This means that for debugging, if you don't have the Oracle 32bit client, you need to use VuGen's Force 16-bit options.

When a new window is created, the application retrieves an .xpf file from the file system for display. Currently, VuGen does not take this into consideration since it records at the client/server level. Therefore, there is a fairly significant inaccuracy in performance measurements since in most cases performance problems are related to the network bottleneck between clients and file server. We are currently thinking about this problem and how, if at all, to solve it.

Solving Common Problems with Oracle Applications

This section contains a list of common problems that you may encounter while working with Oracle Applications, and suggested solutions.

ORA-20001 and ORA-06512

Errors ORA-20001 and ORA-06512 appear during replay when the `lrd_stmt` contains the pl/sql block: `fnd_signon.audit_responsibility(...)`

This statement fails during replay because the sign-on number is unique for each new connection.

Solution

In order to solve this problem you need to use the new correlation tool for the sign-on number. This is second assigned value in the statement.

After you scan for possible values to correlate, highlight the value of the second `lrd_assign_bind()` for the failed statement. Note that the values in the "correlated query" window may not appear in the same order as the actual recorded statements.

The grid containing the substitution value should appear after the `lrd_stmt` which contains the pl/sql block: `fnd_signon.audit_user(...)`.

Note: Since the sign-on number is unique for every connection, you need to use correlation for each new connection that you record.

Example of Solution

The following statement failed in replay because the second value, "1498224" is the unique sign-on number for every new connection.

```
lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"
           "; end;", -1, 1, 1, 0);
lrd_assign_bind(Csr6, "s", "D", &s_D216, 0, 0, 0);
lrd_assign_bind(Csr6, "l", "1498224", &l_D217, 0, 0, 0);
lrd_assign_bind(Csr6, "f", "1", &f_D218, 0, 0, 0);
lrd_assign_bind(Csr6, "a", "810", &a_D219, 0, 0, 0);
lrd_assign_bind(Csr6, "r", "20675", &r_D220, 0, 0, 0);
lrd_assign_bind(Csr6, "t", "Windows PC", &t_D221, 0, 0, 0);
lrd_assign_bind(Csr6, "p", "", &p_D222, 0, 0, 0);
lrd_exec(Csr6, 1, 0, 0, 0, 0);
```

The sign-on number can be found in the `lrd_stmt` with "`fnd_signon.audit_user`". The value of the first placeholder "a" should be saved. The input of "a" is always "0" but the output is the requested value.

Modified code:

```

lrd_stmt(Csr4, "begin fnd_signon.audit_user(:a,:l,:u,:t,:n,:p,:s); end;", -1, 1, 1,
0);
  lrd_assign_bind(Csr4, "a", "0", &a_D46, 0, 0, 0);
  lrd_assign_bind(Csr4, "l", "D", &l_D47, 0, 0, 0);
  lrd_assign_bind(Csr4, "u", "1001", &u_D48, 0, 0, 0);
  lrd_assign_bind(Csr4, "t", "Windows PC", &t_D49, 0, 0, 0);
  lrd_assign_bind(Csr4, "n", "OraUser", &n_D50, 0, 0, 0);
  lrd_assign_bind(Csr4, "p", "", &p_D51, 0, 0, 0);
  lrd_assign_bind(Csr4, "s", "14157", &s_D52, 0, 0, 0);
  lrd_exec(Csr4, 1, 0, 0, 0, 0);

lrd_save_value(&a_D46, 0, 0, "saved_a_D46");
  Grid0(17);

lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"
"; end;", -1, 1, 1, 0);
  lrd_assign_bind(Csr6, "s", "D", &s_D216, 0, 0, 0);
  lrd_assign_bind(Csr6, "l", "<saved_a_D46>", &l_D217, 0, 0, 0);
  lrd_assign_bind(Csr6, "f", "1", &f_D218, 0, 0, 0);
  lrd_assign_bind(Csr6, "a", "810", &a_D219, 0, 0, 0);
  lrd_assign_bind(Csr6, "r", "20675", &r_D220, 0, 0, 0);
  lrd_assign_bind(Csr6, "t", "Windows PC", &t_D221, 0, 0, 0);
  lrd_assign_bind(Csr6, "p", "", &p_D222, 0, 0, 0);
  lrd_exec(Csr6, 1, 0, 0, 0, 0);

```

Working with large numbers

Large numbers (NUMBER data type) sometimes appear in different format in the GRID and in the ASCII file. This difference makes it more difficult to identify numbers while searching for values to save for correlation.

For example, you could have a value appear as 1000003 in the grid, but as 1e+0006 in the Recording Log (ASCII file).

Workaround

If you have an error during replay and the correlation tool cannot locate the value in previous results, look for this value in the other format in grid.

ORA-00960

This error can occur if the column names in the recorded script are not unique. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "  
              "MTL_UNITS_OF_MEASURE "  
              "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "  
              "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

In this case you receive the following error:

```
"lrd.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960:  
ambiguous column naming in select list".
```

Workaround

Change the statement by adding an alias to at least one of the non-unique columns, thus mapping it to a new unique name. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE second, DESCRIPTION FROM "  
              "MTL_UNITS_OF_MEASURE "  
              "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "  
              "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

Alternate Workaround: remove ORDER BY from the lrd statement

ORA-2002

Error 2002 appears when you try to use an unopened cursor. It occurs when you replay a user more than one iteration and you recorded into more than one section of the script.

Specifically, if a cursor is opened in the *vuser_init* section and closed in the Actions section, then you will encounter this error on the second iteration if you try to use the cursor. This is because it was closed but not re-opened.

For example: You have *lrd_open_cursor* in the *vuser_init* section and *lrd_close_cursor* in the Actions section. If you replay this user more than one iteration, you are going to get an error in the second iteration because you

try using an unopened cursor (it was closed in first iteration, but not re-opened in the second).

Workaround

The easiest way to solve this is to move the `lrd_close_cursor` or/and `lrd_close_connection` of the problem cursor to the `vuser_end` section.

Two-tier Database Scripting Tips

The following section offers solutions for two-tier database scripts. For Siebel specific solutions, see the next section.

Question 1: Why does the script fail when it is data driven, while the same values work with the application itself?

Answer: The failure may be a result of trailing spaces in your data values. Even though the data values that you type directly into the GUI are probably truncated, you should manually eliminate them from your data file. Tab-delimited files can hide trailing spaces and therefore obscure problems. In general, comma-delimited files are recommended. You can view the files in Excel to see if things are correct.

Question 2: Why does an SQL error of an invalid cursor state occur on the second iteration?

Answer: The `lrd_close_cursor` function may not have been generated or it may be in the `end` section instead of the `action` section. You will need to add a cursor close function or move it from the `end` section to make the script iterate successfully.

Opening a new cursor may be costly in terms of resources. Therefore, it is recommended that you only open a cursor once in the `actions` section during the first iteration. You can then add a new parameter that contains the iteration number as a string by using the `Iteration Number` type. Call this parameter `IterationNum`. Then, inside the `actions` section replace a call to open a new cursor like

```
lrd_open_cursor(&Csr1, Con1, 0);
```

with

```
if (!strcmp(lr_eval_string("<IterationNum>"), "1"))  
    lrd_open_cursor(&Csr1, Con1, 0);
```

Question 3:How can I fix code produced by VuGen that will not compile because of data declarations in the *vd.f.h* file?

Answer: The problem, most likely, is an SQL data type that is not supported by VuGen. For Microsoft SQL, you can often work around this issue by replacing the undefined error message in *vd.f.h* with "DT_SZ" (null terminated string). Although this is not the actual datatype, VuGen can compile the script correctly. Please report the problem and send the original script to customer support.

Question 4: What is the meaning of LRD Error 2048?

Answer: VuGen is failing because it is trying to bind a variable with a longer length than what was allocated during recording. You can correct this by enlarging the variable definition in *vd.f.h* to receive a longer string back from the database. Search this file for the unique numeric identifier. You will see its definition and length. The length is the third element in the structure. Increase this length as required and the script will replay successfully. For example, in the following script, we have:

```
lrd_assign(&_2_D354, "<ROW_ID>", 0, 0, 0);
```

In *vd.f.h*, we search for *_2_D354* and find

```
static LRD_VAR_DESC _2_D354 = {  
    LRD_VAR_DESC_EYECAT, 1, 10, LRD_BYTYPE_ODBC,  
    {0, 0, 0}, DT_SZ, 0, 0, 15, 12};
```

We change it to:

```
static LRD_VAR_DESC _2_D354 = {  
    LRD_VAR_DESC_EYECAT, 1, 12, LRD_BYTYPE_ODBC,  
    {0, 0, 0}, DT_SZ, 0, 0, 15, 12};
```

The complete definition of LRD_VAR_DESC appears in *lrd.h*. You can find it by searching for typedef struct LRD_VAR_DESC.

Question 5: How can I obtain the number of rows affected by an UPDATE, INSERT or DELETE when using ODBC and Oracle?

Answer: You can use **lrd** functions to obtain this information. For ODBC, use **lrd_row_count**. The syntax is:

```
int rowcount;
.
.
.
lrd_row_count(Csr33, &rowcount, 0);
```

Note that **lrd_row_count** must immediately follow the pertinent statement execution.

For Oracle you can use the fourth argument of **lrd_exec**.

```
lrd_exec(Csr19, 1, 0, &rowcount, 0, 0);
```

If you are using Oracle's OCI 8, you can use the fifth argument of **lrd_ora8_exec**.

```
lrd_ora8_exec(OraSvc1, OraStm3, 1, 0, &uliRowsProcessed, 0, 0, 0, 0);
```

Question 6: How can I avoid duplicate key violations?

Answer: Occasionally, you will see a duplicate key violation when performing an Insert. You should be able to find the primary key by comparing two recordings to determine the problem. Check whether this or earlier UPDATE or INSERT statement should use correlated queries. You can use the data dictionary in order to find the columns that are used in the violated unique constraint.

In Oracle you will see the following message when a unique constraint is violated:

```
ORA-00001: unique constraint (SCOTT.PK_EMP) violated
```

In this example SCOTT is the owner of the related unique index, and PK_EMP is the name of this index. Use SQL*Plus to query the data dictionary to find the columns. The pattern for this query is:

```
select column_name from all_ind_columns where index_name = '<IndexName>'
and index_owner = '<IndexOwner>';
```

```
select column_name from all_ind_columns where index_name = 'PK_EMP' and
index_owner = 'SCOTT';
```

Since the values inserted into the database are new, they might not appear in earlier queries, but they could be related to the results of earlier queries, such as one more than the value returned in an earlier query.

For Microsoft SQL Server you will see one of these messages:

```
Cannot insert duplicate key row in object 'newtab' with unique index
'IX_newtab'.
```

```
Violation of UNIQUE KEY constraint 'IX_Mark_Table'. Cannot insert duplicate key
in object 'Mark_Table'.
```

```
Violation of PRIMARY KEY constraint 'PK_NewTab'. Cannot insert duplicate key in
object 'NewTab'.
```

You can use the Query Analyzer to find out which columns used by the key or index. The pattern for this query is:

```
select C.name
      from sysindexes A, sysindexkeys B, syscolumns C
      where C.colid = B.colid and C.id = B.id and
      A.id = B.id and A.indid = B.indid
      and A.name = '<IndexName>' and A.id = object_id('<TableName>')
```

```
select C.name
      from sysindexes A, sysindexkeys B, syscolumns C
      where C.colid = B.colid and C.id = B.id and
      A.id = B.id and A.indid = B.indid
      and A.name = 'IX_newtab' and A.id = object_id('newtab')
```

For DB2 you might see the following message:

```
SQL0803N One or more values in the INSERT statement, UPDATE statement, or
foreign key update caused by a DELETE statement are not valid because they
would produce duplicate rows for a table with a primary key, unique constraint,
or unique index. SQLSTATE=23505
```

If you still encounter problems, be sure to check the number of rows changed for Updates and Inserts for both recording and replay. Very often, an UPDATE fails to change any rows during replay, because the WHERE clause was not satisfied. This does not directly result in an error, but it causes a table not to be properly updated, and can cause a later SELECT to choose the wrong value when correlating the query.

Also verify that there are no problems during multi-user replay. In certain instances, only one user will successfully perform an UPDATE. This occurs with Siebel, where it is necessary to manually write a loop to overcome the problem.

Question 7: The database does not appear to be modified after replaying a script which should have modified the database.

Answer: Through the user application's UI, check if the updated values appear when trying to see the current data accessible to the application. If the values have not been updated, you need to determine they were not changed. Possibly, an UPDATE statement changed one or more rows when the application was recorded, and did not change any during replay.

Check these items:

- **Verify statement:** If there is a WHERE clause in the UPDATE statement, verify that it is correct.
- **Check for correlations:** Record the application twice and compare the UPDATE statements from each of the recordings to make sure that the necessary correlations were performed.
- **Check the total number of rows:** Check the number of rows that were changed after the UPDATE. For Oracle, this information is stored in the fourth parameter of `lrd_exec`. For ODBC, use `lrd_row_count` to determine the number of rows updated. You can also add code to your script that prints the number of rows that were updated. If this value is 0, the UPDATE failed to modify the database.
- **Check the SET clause:** Check the SET clause of the UPDATE statement. Make sure that you correlated any necessary values here instead of hard-coding them. You can see this by comparing two recordings of the UPDATE.

In certain cases, the UPDATE works when replaying one Vuser, but not for multiple Vusers. The UPDATE of one Vuser might interfere with that of another. Parameterize each Vuser so that each one uses different values during the UPDATE, unless you want each vuser to update with the same values. In this case try adding retry logic to perform the UPDATE a second time.

Question 8: How do I avoid the unique column name error when replaying a statement recorded with an Oracle Application. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "
             "MTL_UNITS_OF_MEASURE "
             "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
             "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

The following error message was issued:

```
"lrd.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960:
ambiguous column naming in select list".
```

Answer: Change the statement by adding an alias to at least one of the non-unique columns, thereby mapping it to a new unique name. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE second, DESCRIPTION FROM"
             "MTL_UNITS_OF_MEASURE "
             "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
             "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

Siebel-specific Scripting Tips

This section offers solutions for Siebel database users. You should also refer to the previous section which discusses some general database scripting tips.

Question 9: Virtual users run fine in VuGen but fail in the Console with duplicate key violations.

Answer: The Siebel client stores a key in the NEXT_SUFFIX column of the S_SSA_ID table. This client has code that detects and recovers from situations in which it fails to successfully get a block of suffix values.

ProTune automatically correlates the NEXT_SUFFIX and MODIFICATION_NUM fields of the S_SSA_ID table. During an UPDATE the MODIFICATION_NUM field is incremented by 1 and the NEXT_SUFFIX

field is increased by 100 in base 36. However, ProTune does not add code in instances where a client could not obtain a new block of suffix values. As a result, the replay fails with a unique constraint error, when you attempt to insert new values into the database.

You must manually add code to each location in the script where a block of suffixes is obtained, in order to perform a retry if the first attempt fails. You can locate these places by searching for `SiebelPreSave` in the script. You must also add a `while` loop with code similar to the example below. This example only works for Oracle. For ODBC use `lrd_row_count` instead of using the fourth argument of `lrd_exec`.

```
unsigned long lRowUpdated;
int nAttempt;

...

// This loops until we successfully obtain a "next_suffix"
lRowUpdated = 0;
nAttempt=0;

while (lRowUpdated != 1) {

    nAttempt++;
    if (nAttempt > 1)
        lrd_output_message (".....Next suffix retry %d", nAttempt);
    else
    {
        lrd_open_cursor(&Csr13, Con1, 0);
        lrd_stmt(Csr13, "SELECT\n T1.LAST_UPD,\n T1.CREATED_BY,\n "
            "T1.CONFLICT_ID,\n T1.CREATED,\n T1.NEXT_SUFFIX,\n "
            "T1.ROW_ID,\n T1.NEXT_PREFIX,\n T1.CORPORATE_PREFIX,\n "
            "T1.MODIFICATION_NUM,\n T1.NEXT_FILE_SUFFIX,\n "
            "T1.LAST_UPD_BY\n FROM \n SIEBEL.S_SSA_ID T1", -1, 1, 1, 0);
    }
    lrd_bind_cols(Csr13, BCInfo_D375, 0);
    lrd_exec(Csr13, 0, 0, 0, 0, 0);

    SiebelPreSave_1();
    lrd_fetch(Csr13, -1, 4, 0, PrintRow26, 0);
```

```

GRID(26);
SiebelPostSave_1();

if (nAttempt > 1)
{
    lrd_open_cursor(&Csr14, Con1, 0);
    lrd_stmt(Csr14, "\nUPDATE SIEBEL.S_SSA_ID SET\n LAST_UPD_BY=:1,\n "
        "NEXT_SUFFIX = :2,\n    MODIFICATION_NUM = :3,\n    LAST_UPD = "
        ":4\n WHERE\n ROW_ID = :5 AND MODIFICATION_NUM = :6\n", -1, 1,
        1, 0);
}
lrd_assign_bind(Csr14, "6", "<modification_num>", &_6_D376, 0,
    LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "5", "0-11", &_5_D377, 0, LRD_BIND_BY_NUMBER, 0);
strcpy (szTimeAtNewButton, lr_eval_string("<Now>"));
sprintf (szTimeStamp, "%s %s", lr_eval_string("<Today>"),
    szTimeAtNewButton);
lr_save_string (szTimeStamp, "DateTimeStamp");
lrd_assign_bind(Csr14, "4", "<DateTimeStamp>", &_4_D378, 0,
    LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "3", "<next_modnum>", &_3_D379, 0,
    LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "2", "<next_suffix_x100>", &_2_D380, 0,
    LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "1", "1-1E1", &_1_D381, 0, LRD_BIND_BY_NUMBER, 0);

// this update won't update any rows unless we successfully got our suffix
lrd_exec(Csr14, 1, 0, &lRowUpdated, 0, 0);
lrd_commit(0, Con1, 0);

} // while
    lr_output_message ("...Rows updated %ld", lRowUpdated);

```

Question 10: How can I find the correct value to correlate for a primary key?

Answer: Siebel tends to generate key values based on base 36 mathematical manipulations of *<next_suffix>*. Try comparing several recordings and try to

determine the relationships. You can ignore date fields when correlating Siebel, since they do not seem to effect script replay.

Question 11: How can I solve an INSERT into S_SRV_REQ failure with a duplicate key violation?

Answer: The primary key is SR_NUM. Newer versions of ProTune should automatically correlate insertions into this table, by using the function `lrd_siebel_str2num`, which converts the NEXT_SUFFIX value of the S_SSA_ID table from base 36 to the base 10 equivalent. Older versions of ProTune might not handle this correlation correctly.

Question 12: ProTune does not automatically perform all the correlations I need in order to replay my script correctly. How can I add the missing correlations?

Answer: Currently ProTune only saves the values of the NEXT_SUFFIX and MODIFICATION_NUM columns from the S_SSA_ID table and replaces them with parameters when they are used later in the script. You may need to add some additional correlations manually. The correlation code in the **SiebelPreSave** and **SiebelPostSave** functions in the *print.inl* file can serve as an example of how to correlate specific values once you determine what needs to be correlated.

- Sometimes the NEXT_FILE_SUFFIX and MODIFICATION_NUM columns are chosen from the S_SSA_ID table. In this case, an UPDATE statement updates the NEXT_FILE_SUFFIX by adding one to this string in base 36, and one to the MODIFICATION_NUM. The value of the NEXT_FILE_SUFFIX will often be inserted in the FILE_REV_NUM field of a table. Often the name of this table ends with the *_ATT* suffix, to indicate that it is an attachment.
- Whenever Siebel performs an UPDATE statement, there is a MODIFICATION_NUM column that is incremented by one. ProTune only generates this correlation automatically for the S_SSA_ID table. You have to do it manually for other cases.
- Siebel refers to records according to their ID number. Siebel usually finds all records of a particular type (such as an agreement), and then later uses the ID number for a record when trying to update or delete an existing record of this type. You need to replace the ID number by a parameter

during replay in order to generate a meaningful load test. The ID number has the form of one or more digits, a hyphen, followed by one or more alphanumeric characters, such as 1-QPF9. ProTune does not do this parameterization automatically, so you have to do it manually.

- If you find any other missing correlations or parameterizations, please notify customer support in order that Mercury Interactive can improve ProTune's support for Siebel.

65

Advanced Topics

This chapter contains additional information for advanced ProTune users.

- ▶ Files Generated During Recording
- ▶ Files Generated During Replay
- ▶ Specifying the Vuser Behavior
- ▶ Command Line Parameters
- ▶ Examining the .dat Files
- ▶ Adding a New Vuser Type

Files Generated During Recording

Assume that the recorded test has been given the name ‘vuser’ and is stored under c:\tmp. Following is a list of the more important files that are generated after recording:

vuser.usr	Contains information about the virtual user: type, AUT, action files, and so forth.
vuser.bak	A copy of Vuser.usr before the last save operation.
default.cfg	Contains a listing of all run-time settings as defined in the VuGen application (think time, iterations, log, web).
vuser.asc	The original recorded API calls.
vuser.grd	Contains the column headers for grids in database scripts.

default.usp	Contains the script's run logic, including how the actions sections run.
init.c	Exact copy of the Vuser_init function as seen in the VuGen main window.
run.c	Exact copy of the Action function as seen in the VuGen main window.
end.c	Exact copy of the Vuser_end function as seen in the VuGen main window.
vdf.h	A header file of C variable definitions used in the script.
\Data	The Data directory stores all of the recorded data used primarily as a backup. Once the data is in this directory, it is not touched or used. For example, Vuser.c is a copy of run.c .

Example of Vuser.usr File

```
[General]
Type=Oracle_NCA
DefaultCfg=default.cfg
AppName=C:\PROGRA~1\Netscape\COMMUN~1\Program\netscape.exe
BuildTarget=
ParamRightBrace=>
ParamLeftBrace=<
NewFunctionHeader=0
MajorVersion=5
MinorVersion=0
ParameterFile=nca_test3.prm
GlobalParameterFile=
[Transactions]
Connect=
[Actions]
vuser_init=init.c
Actions=run.c
vuser_end=end.c
```

Example of default.cfg File

```
[General]
XlBridgeTimeout=120

[ThinkTime]
Options=NOTHINK
Factor=1
LimitFlag=0
Limit=1

[Iterations]
NumOfIterations=1
IterationPace=IterationASAP
StartEvery=60
RandomMin=60
RandomMax=90

[Log]
LogOptions=LogBrief
MsgClassData=0
MsgClassParameters=0
MsgClassFull=0
```

Files Generated During Replay

This section describes what occurs when the Vuser is replayed.

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

Note: The patch **cpp.exe** for beta2 is a totally different shareware executable than the earlier version, which was very problematic.

The following command line is used:

```
cpp -foptions.txt
```

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

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

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

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

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

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

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

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

Example of options.txt file

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

Example of Vuser.c file

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

Specifying the Vuser Behavior

Since VuGen creates the Vuser script and the Vuser behavior as two independent sources, you can configure user behavior without directly referencing the Vuser script, for example, wait times, pacing times, looping iterations, logging, and so forth. This means that it is very easy to make configuration changes to a Vuser, as well as store several such 'profiles' for the same Vuser script.

The '*Vuser.cfg*' file, by default, is responsible for defining this behavior - as specified in VuGen's Runtime settings dialog box. You can save several versions of this file for different user behavior and then run the Vuser script referencing the relevant *.cfg* file.

Command Line Parameters

The Vusers can accept command line parameters when invoked. There are several VuGen functions available to reference them (`lr_get_attrib_double` etc.).

When running the Vuser from VuGen, you cannot control the command line parameters. You can do this manually however from the Windows command line by adding the parameters at the end of the line, after all the other driver parameters, for example:

```
mdrv.exe -usr c:\tmp\Vuser\Vuser.usr -out c:\tmp\vuser  
vuser_command_line_params
```

Examining the .dat Files

There are two .dat files used by VuGen: `vugen.dat` and `mdrv.dat`.

vugen.dat

This `vugen.dat` file resides in the `M_LROOT\dat` directory and contains general information about VuGen, to be used by both the VuGen and the Console.

```
[Templates]
```

```
RelativeDirectory=template
```

The **Templates** section indicates where the templates are for the VuGen protocols. The default entry indicates that they are in the relative *template* directory. Each protocol has a subdirectory under *template*, which contains the template files for that protocol.

The next section is the **GlobalFiles** section.

```
[GlobalFiles]
```

```
main.c=main.c
```

```
@@TestName@@.usr=test.usr
```

```
default.cfg=test.cfg
```

```
default.usp=test.usp
```

The **GlobalFiles** section contains a list of files that VuGen copies to the test directory whenever you create a new test. For example, if you have a test called "user1", then VuGen will copy *main.c*, *user1.usr* and *user1.cfg* to the test directory.

The **ActionFiles** section contains the name of the file containing the Actions to be performed by the Vuser and upon which to perform iterations.

```
[ActionFiles]
@@actionFile@@=action.c
```

In addition to the settings shown above, *vugen.dat* contains settings that indicate the operating system and other compilation related settings.

mdrv.dat

The *mdrv.dat* file contains a separate section for each protocol defining the location of the library files and driver executables. The next section describes what you need to add to this file in order to define a new protocol.

Adding a New Vuser Type

To add a new Vuser type/protocol to VuGen, you need to:

- Edit the *mdrv.dat* file with the new protocol's settings.
- Add a *.cfg* file
- Insert an *.lrp* file.
- Create a Template Directory

Editing the mdrv.dat File

First, you edit the *mdrv.dat* file which resides in the *M_LROOT\dat* directory. You add a section for the new Vuser type with all of the applicable parameters from the following list.

```
[<extension_name>]
ExtPriorityType=< {internal, protocol}>
WINNT_EXT_LIBS=<dll name for NT>
```

WIN95_EXT_LIBS=<dll name for 95>
SOLARIS_EXT_LIBS=<dll name for Solaris>
LINUX_EXT_LIBS=<dll name for Linux>
HPUX_EXT_LIBS=<dll name for HP>
AIX_EXT_LIBS=<dll name for IBM>
LibCfgFunc=<configuration function name>
UtilityExt=<other extensions list>
WINNT_DLLS=<dlls to load to the interpreter context, for NT>
WIN95_DLLS=<dlls to load to the interpreter context, for 95>
SOLARIS_DLLS=<dlls to load to the interpreter context, for Solaris>
LINUX_DLLS=<dlls to load to the interpreter context, for Linux>
HPUX_DLLS=<dlls to load to the interpreter context, for HP>
AIX_DLLS=<dlls to load to the interpreter context, for IBM>
ExtIncludeFiles=<extra include files. several files can be separated by a comma>
ExtCmdLineConc=<extra command line (if the attr exists concatenate value)>
ExtCmdLineOverwrite=<extra command line (if the attr exists overwrite value)>
CallActionByNameFunc=<interpreter exec_action function>
GetFuncAddress=<interpreter get_location function>
RunLogicInitFunc=<action_logic init function>
RunLogicRunFunc=<action_logic run function>
RunLogicEndFunc=<action_logic end function>

For example, an Oracle NCA Vuser type is represented by:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp11i.dll
WIN95_EXT_LIBS=ncarp11i.dll
LINUX_EXT_LIBS=liboranca11i.so
SOLARIS_EXT_LIBS=liboranca11i.so
HPUX_EXT_LIBS=liboranca11i.sl
AIX_EXT_LIBS=liboranca11i.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api,HttpEngine
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
SecurityRequirementsFiles=oracle_nca.asl
SecurityMode=On
```


VuGen was designed to be able to handle a new Vuser type with no code modifications. You may, however, need to add a special View.

There is no generic driver supplied with VuGen, but you can customize one of the existing drivers. To use a customized driver, modify *mdrv.dat*. Add a line with the platform and existing driver, then add a new line with your customized driver name, in the format `<platform>_DLLS=<my_replay.dll name>`. For example, if your SAP replay dll is called SAPPLAY32.DLL, add the following two lines to the [sap] section of *mdrv.dat*:

```
WINNT=sapdrv32.exe
WINNT_DLLS=sapplay32.dll
```

Adding a CFG file

You can optionally specify a configuration file to set the default Run-Time Settings and Recording options for your protocol. You define it in the **LibCfgFunc** variable in the *mdrv.dat* file, or place one called *default.cfg* in the new protocols subdirectory under templates. A sample default.cfg follows.

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

[Iterations]
NumOfIterations=1
IterationPace=IterationASAP
StartEvery=60
RandomMin=60
RandomMax=90

[Log]
LogOptions=LogExtended
MsgClassData=0
MsgClassParameters=0
MsgClassFull=1
```

Inserting an LRP file

In the `dat/protocols` directory, insert an *lrp* file which defines the protocol. This file contains the configuration information for the protocol in the Protocol, Template, Vugen, and API sections. Certain protocols may have additional sections, corresponding to the additional run-time setting options.

The **Protocol** section contains the name, category, description, and bitmap location for the protocol.

```
[Protocol]
Name=WAP
CommonName=WAP
Category=Wireless
Description=Wireless Application Protocol - used for Web-based, wireless
communication between mobile devices and content providers.
Icon=bitmaps\wap.bmp
Hidden=0
Single=1
Multi=0
```

The **Template** section indicates the name of the various sections of the script and the default test name.

```
[Template]
vuser_init.c=init.c
vuser_end.c=end.c
Action1.c=action.c
Default.usp=test.usp
@@TestName@@.usr=wap.usr
default.cfg=default.cfg
```

The **Vugen** section has information about the record and replay engines, along with the necessary DLLs and run-time files.

The **API** section contains information for internal macros.

You can use one of the existing *lrp* files in the protocols directory as a base for your new protocol.

Specifying a Template

After adding an *lrp* file, insert a subdirectory to *M_LROOT*/template with a name corresponding to the protocol name defined in the *lrp* file. In this subdirectory, insert a *default.cfg* file which defines the default settings for the general and run-time settings.

If you want to use a global header file for all of your protocol's scripts, add a file named *globals.h*. This file should contain an include statement which points to a header file for the new protocol. For example, the *template/http* subdirectory contains a file called *globals.h* which directs VuGen to the *as_web.h* file in the include directory:

```
#include #as_web.h"
```


Part XVII

Appendixes

A

The Java Environment: A Comprehensive Guide

This chapter discusses the Java environment. It explains terms that are necessary in order to understand and configure an existing Java environment for certain Java applications. Although the Java language is cross platform, this document only describes issues for Windows NT.

- Terminology
- JDK Versions
- Browsers
- Java Plug-In
- Other Environments
- Frequently Asked Questions

About the Java Environment

The Java programming language is a high-level object-oriented language. The language is known to be simple to develop, distributed, interpreted, secure, portable, and multi-threaded.

Although Java may appear simple for developers, the evolving technology surrounding it causes the environment settings to be complex. SUN, the developer of Java, continues to release new versions of Java, APIs, and tools, making the Java world less manageable.

Terminology

JDK: Java Development Kit. A software development environment developed by SUN Microsystems for producing Java programs. Each release of the JDK contains the Java Compiler, the Java Interpreter, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools. The JDK cannot be redistributed.

JRE: Java Runtime Environment. A subset of the JDK for end-users and developers who want to redistribute the runtime environment. The JRE consists of the Java Virtual Machine, the core classes, and supporting files.

JVM: Java Virtual Machine. The part of the Java runtime environment responsible for interpreting bytecode.

CLASSPATH: An environment variable that tells the Java Virtual Machine where to find the class files. It should include the JDK core libraries and the class libraries used by the Java program. The classpath variable is constructed from a set of entries separated by semicolons. Each entry can be a directory or an archive name (jar or zip files). The current directory '.' should also be included. In JDK 1.1.x the classpath variable should also contain `<JDK>\lib\classes.zip`. The classes are searched for in the CLASSPATH in the order of entry.

PATH: An environment variable containing a list of directories, in which the operating system looks for executable files if it is unable to find the file in the working directory. Use semicolons to separate the directory names. When working with the JDK, `<JDK>\bin` should be placed in the PATH variable in order to make the JDK tools (*java.exe*, *javac.exe*, *appletviewer.exe*, etc.) available.

JAR: Java Archive. A file format used to bundle all components and resources required by a Java applet or application (class files, images, sounds, etc.). In addition, JAR supports data compression, which further decreases download times. By convention, JAR files end with a *.jar* extension. Jar files can be viewed and extracted with the *jar.exe* tool, which comes with the JDK installation, or by using the standard WinZip tool.

JIT: Just-In-Time Compiler. A JIT compiler is an alternative to using a standard Java interpreter. It converts all of the bytecode into native machine code 'just in time'- when the Java program is run. This results in run-time

speed improvements over code that is interpreted by a Java Virtual Machine. This is now included with all versions of JVMs and browsers. In some cases you may want to disable the JIT compiler, e.g., when you want to obtain line numbers in a stack trace instead of “(Compiled Code)” printing, or when you suspect that the JIT is causing bugs.

Applet: A Java program that can be included in an HTML page, and run in the context of a Java-capable browser or the Applet viewer. A security manager restricts the actions of Java applets.

Application: A stand-alone executable program. It has no security or I/O restrictions (unless specifically applied). A Java application requires an interpreter to execute.

bytecode: Machine-independent code generated by the Java compiler and executed by the Java interpreter.

class file: A file containing machine-independent Java byte codes. The Java compiler generates *.class files for the Java interpreter to read.

Exception: An event that occurs during program execution which prevents the program from continuing its normal flow. Generally, this is an unexpected error, followed by a printout of the stack trace to locate the exact place in the code that the exception occurred. The programmer can use the *try*, *catch*, and *throw* keywords to create an exception handler that reacts to a specific type of exception. The programmer can choose to resume executing after the exception handler has executed.

Garbage Collection: The automatic detection and freeing of memory that is no longer in use. The Java runtime system performs garbage collection so that programmers never have to explicitly free objects.

java.exe: The java interpreter that comes with the SUN's JDK.

appletviewer.exe: A tool for running applets without a browser. It comes with SUN Microsystems' JDK. The Appletviewer receives an HTML page as an argument and searches it for an <Applet> tag. It then loads this applet.

javap.exe: A tool that disassembles Java .class files. It comes with SUN's JDK. This tool is useful for verifying API classes in the absence of sources.

javac.exe: The Java compiler that is provided with SUN's JDK.

jview.exe: Microsoft's Java Virtual Machine. It is integrated in the Internet Explorer.

HTML Attribute: HTML pages that load Java Applets must include an `<Applet>` tag that specifies details about the name, the place, and other properties of the applet. The three attributes in this tag that give a complete specification of where to find the main applet class files are:

The *code* attribute specifies the name of the main applet class file.

The *codebase* attribute specifies the URL of the directory containing the file, and other files.

The *archives* attribute specifies any additional archives: *jar/zip/cab* files that contain classes and resource files. These archives are placed under the codebase.

classes.zip: The zip file that contains all the Java core classes of JDK1.1.x. It can be found under `<JDK>\lib\classes.zip`. These classes are essential for running any Java application and must be placed in CLASSPATH.

rt.jar: The jar file that contains all the Java core classes of JDK1.2.x and JDK1.3. It can be found under `<JDK>\jre\lib\rt.jar`. This jar does *not* need to be in the CLASSPATH, because it is retrieved automatically by the JVM. However, if the Xboot classpath (see below) is replaced, make sure *rt.jar* is present.

JavaScript: An extension to HTML that allows you to incorporate some dynamic functionality on a Web page. Do not confuse this term with the *Java* language. JavaScript is usually interpreted by a special mechanism in the browser that has nothing to do with the Java language Virtual Machine. However, it is possible to call from within JavaScript to static methods in Java's standard JDK. This will eventually cause the loading of the JVM and its DLLs.

JDK Versions

The following section describes the various versions of the Java Development Kit, JDK.

JDK 1.1

The following are the command line arguments for *java.exe* executable file in JDK1.1:

Command Line Argument	Description
<i>-version</i>	prints the exact build version
<i>-verbose</i>	turns on verbose mode and produces printouts from the system class loader about the name and path of each class being loaded.
<i>-noasyncgc</i>	disables asynchronous garbage collection
<i>-verbosegc</i>	prints a message when garbage collection occurs
<i>-noclassgc</i>	disables class garbage collection
<i>-ms<number></i>	sets the initial Java heap size. For example: <i>-ms256M</i>
<i>-mx<number></i>	sets the maximum Java heap size
<i>-classpath <dirs separated by ;></i>	lists directories in which to look for classes instead of the CLASSPATH variable.
<i>-noverify</i>	does not verify any class
<i>-nojit</i>	disables the JIT compiler
<i>-D<name>=<value></i>	sets a system property

JDK1.1 versions:

The first generation of JDK, versions 1.0.1 and 1.0.2, used the old event model. The JDK1.1 generation introduced the new event model. Today, older versions through JDK1.15 are not supported officially by SUN.

The supported versions are: 1.1.6, 1.1.7a, 1.1.7b and 1.1.8. JDK1.1.6 is known to be unstable. The other versions have slight differences among them, mainly regarding security fixes.

JDK 1.1 Examples:

The following example demonstrates how to run an application named TestApp with no JIT compiler, and to take the classes from jdk and the current directory.

```
java -nojit -classpath d:/jdk1.1.7b/lib/classes.zip;. TestApp
```

The following example demonstrates running the TestApp application with Java initial, and with maximum heap size set to 256M. The Test application has two arguments. Classes are taken from the global CLASSPATH variable.

```
java -ms256M -mx256M TestApp apparg1 apparg2
```

Example 3

The following example demonstrates checking the current Java version placed in PATH:

```
java -version
```

JDK 1.2

In JDK1.1.x, the search for classes was done according to the CLASSPATH environment variable. For applets, the search for classes was also done in the codebase and archives specified in HTML. In JDK1.2.x, a new element determines the place of classes: the 'bootstrap classpath'. The *bootstrap classpath* precedes the classpath. By default, it contains the JDK core classes, placed under `<JDK>\jre\lib\rt.jar`. The bootstrap classpath can be changed using the `-Xbootclasspath` command line variable. In addition, *javac* supports a similar option ('-bootclasspath') which can be used to change the platform classes you compile.

JDK1.2.x includes the classes of the JFC1.1 library, and Java IDL CORBA classes. All are placed in the system's *rt.jar* file.

In JDK1.2 or higher there are two types of virtual machines: the *Classic* virtual machine, which is located under `<JDK>/jre/bin/classic/jvm.dll`, and the *HotSpot* virtual machine, which is located under

`<JDK>/jre/bin/hotspot/jvm.dll`. The HotSpot technology produces superior performance compared to that of the Classic VM. To use the Classic VM, use the `-classic` java command-line option.

In JDK1.2, you can run a Java application by specifying only a jar name: `java.exe -jar <jar file name>`. In this case, the JVM opens the jar and searches for a manifest file (Manifest.mf). This file should specify the name of the class containing the main method. The JVM then loads that specific class and runs the main method.

JDK 1.2 also has improvements in functionality, performance, security, and global support over previous versions of the Java platform.

The following table shows the non-standard command line options for `java.exe` in JDK 1.2.

Command Line Argument	Description
<code>-Xbootclasspath:<dirs and zip/jar files separated by ;></code>	sets the search path for bootstrap classes and resources
<code>-Xbootclasspath/p:<dirs and zip/jar files separated by ;></code>	prepend the default search path for bootstrap classes and resources
<code>-Xbootclasspath/a:<dirs and zip/jar files separated by ;></code>	appends the default search path for bootstrap classes and resources
<code>-Xnoclassgc</code>	disables class garbage collection
<code>-Xms<size></code>	sets the initial Java heap size. For example: <code>-Xms128M</code>
<code>-Xmx<size></code>	sets maximum Java heap size
<code>-Xrunhprof[:help][[:<option>=<value>, ...]</code>	performs heap, CPU, or monitor profiling

Usage of `-Xbootclasspath`

The `_Xbootclasspath` parameter is very powerful, and should be used with caution. Make sure that the appropriate Java platform classes are being used. These classes are placed in `<JDK>\jre\lib\rt.jar`.

If you use `-Xbootclasspath:<...>`, you must specify the full path to the `rt.jar` file. If you do not, you will not be able to launch the VM. You also cannot specify classes that are part of a JDK of another version in this parameter. If you use the prepend (`/p`) or append (`/a`) format, you can omit the `rt.jar` path.

The syntax of the parameter is similar to the use of the CLASSPATH environment variable. Quotes should be used when an entry includes spaces.

Installing JDK1.2.x

The installation of JDK1.2.x requires places the java.exe file under the <Winnt>\system32 directory, so that the user is not obligated to place the <JDK>\bin directory in the PATH variable. This should be noted especially when working with other JDK versions. The <Winnt>\system32 directory is in the PATH variable and may appear prior to other <JDK>\bin directories of other versions. This is very likely to cause unexpected runtime conflicts (runtime classes that don't match the JVM) when switching between JDK versions.

JDK1.2 versions:

The existing JDK1.2 versions are 1.2, 1.2.1, and 1.2.2. There are only slight differences between them.

JDK1.2 examples:

The following example illustrates running the TestApp application and using the ProTune classes before the JDK1.2.2 Java platform classes. The two statements perform the same task:

```
java -Xbootclasspath:c:\ProTune\classes;c:\jdk1.2.2\jre\lib\rt.jar TestApp
java -Xbootclasspath/p:c:\ProTune\classes TestApp
```

The following example illustrates running the TestApp application with initial and maximum Java heap size set to 256M. Use the -D parameter to set a system property in the Java application:

```
java -Xms256M -Xmx256M -Dtestprop=propvalue TestApp
```

JDK 1.3

JDK1.3 differs from previous versions, primarily in its Virtual Machine (VM). JDK1.3 is shipped with the Java HotSpot Client VM as their default VM implementation. To switch to the Classic VM, use the *-classic* option. The

Java plug-in 1.3 VM, only provides the HotSpot VM. To use the Classic VM in the plug-in, copy the `<JDK>/jre/bin/classic` directory from the JDK directory into the `<plug-in>/bin` directory, and set the `-classic` flag in the control panel.

JNDI classes and RMI/IIOP are included in the `rt.jar` file.

This version contains Improvements in functionality, performance, GUI, security and global support over previous versions of the Java platform.

JDK1.3 versions:

The JDK1.3 existing versions are 1.3, 1.3.0_01. There are slight differences between them.

JDK1.3 Examples

The following example illustrates running the TestApp application with no JIT compiler, and with verbose printouts about classes source directories.

```
java -verbose -Djava.compiler=NONE TestApp
```

The following example illustrates running the TestApp application with the Classic VM TestApp:

```
java -classic TestApp
```

Browsers

The following section describes how to use Netscape and Internet Explorer with java applets or applications.

Appletviewer

Appletviewer is a command-line tool that enables running Java applets without a browser. It accepts a URL address (HTML or file) with an `<Applet>` tag in it.

The Appletviewer tool supports most of the java.exe command-line parameters, but to use them, you must use a '-J' prefix. For example: *-J-verbose*, *-J-Xbootclasspath:<...>*, etc.

Appletviewer Examples

The following example shows the running of an applet that is referenced from an HTML file on the Web.

```
appletviewer http://www.apptest.com/test/test.html
```

The following example shows the running of an applet that is referenced from a local HTML file, and prepending classes to rt.jar (JDK1.2 and higher only).

```
appletviewer -J-Xbootclasspath/p:c:\ProTune\classes  
file:c:\apptest\test.html
```

The following example shows the running of an applet that is referenced from an HTML file in the current directory, with classes displayed in verbose printouts, and with specified maximum and minimum heap size (JDK1.1 style).

```
appletviewer -J-verbose -J-ms256M -J-mx256M test.html
```

Internet Explorer

Java Console

The output messages sent by a Java applet to *stdout* or *stderr* are printed to the Internet Explorer Java console. You open the Java Console by using the [View] menu in your browser.

If you do not see the Java Console option in your browser, then you must enable the Java Console option in the Internet Explorer configuration. To do this, right-click the Internet Explorer icon and choose the 'Properties' option. Then, in the 'Advanced' tab, locate the 'Java VM' section. Select the 'Java console enabled' check box, and restart the browser.

Internet Explorer Java Virtual Machine version

One of the components installed in your Internet Explorer is the Microsoft Java VM. The version of the JVM installed in your browser is independent and different from your browser's version. It is specified in the first line of the Java Console. For example, 'Microsoft (R) VM for Java, 5.0 Release 5.0.0.3176' means that you have a JVM that supports IE5, and whose specific build number is 3176.

You can choose to update just the JVM component in your Internet Explorer to work with a newer build, but you cannot downgrade it.

Internet Explorer Java classes

Internet Explorer's JVM has its own system classes. These classes can be found in the zip files placed under the `<Winnt>\Java\Packages` directory. All of these zip files are inserted automatically in the CLASSPATH by the browser.

Some applets may download zip files and place them under the browser directories. These zip files may contain classes that cause conflicts.

Netscape

Java Console

The output messages sent by a Java applet to *stdout* or *stderr* are printed on the Netscape Java console. To open the Java Console, Choose **Communicator > Tools > Java Console**.

The Java console offers several options that can be activated by typing a certain letter on your keyboard. All options can be viewed by typing '?'. The options include:

- b: Break into the debugger (Windows only)
- c: Clear console window
- d: Dump applet context state to console
- f: Finalize objects on finalization queue
- g: Garbage collect
- h: Print this help message
- l: Capture all classes loaded by an applet to a directory
- m: Print current memory use to console

q: Hide console
s: Dump memory summary to "memory.out"
t: Dump thread info to "memory.out"
x: Dump memory to "memory.out"
X: Dump memory (detailed) to "memory.out"
0-9: Set applet debug level to <n>

Note that the console refreshes frequently, making it difficult to copy text from it. If you select a block of text and the browser sends a printout, it will replace the selected text. Try to wait until the browser has nothing else to send.

Netscape Java classes

The Netscape installation comes with its own JVM classes. These classes can be found in all the jar files located under the <Netscape folder>\Communicator\Programs\java\classes directory. The main jar containing the Java system classes is called java40.jar. Netscape also contains CORBA Visigenic3.0 classes in a jar file called *iiop10.jar*. Occasionally, this jar file causes conflicts in CORBA applications that use other CORBA classes. The user may therefore choose to remove it from this directory. All the jar files are inserted automatically to the CLASSPATH by the browser.

Note that some environments may include additional jars, which do not come with Netscape, but rather were placed there by other installations. These jars may contain classes that cause conflicts. The verbose flag (debug level: 0-9) can be analyzed to understand the source of a particular class.

Java Plug-In

The Java Plug-in is a tool that enables you to run Java Applets in Internet Explorer or Netscape using SUN's JRE, and not using the browser's internal JVM. The first time the Web browser encounters a Web page that specifies the use of the Java Plug-in, the browser must download and install the required files. In subsequent encounters of Web pages that specify the use of the Java Plug-in, it is invoked instantaneously from the user's hard drive and the applet is rendered using SUN's installed JRE.

The Java Plug-in Control Panel

The Java Plug-in comes with a Control Panel that enables you to configure some options. Open the Control Panel from **Start Menu > Programs > The Java Plug-in Control Panel**.

The configurable options include the following: enabling a Java Console, using command-line parameters, changing the version of SUN's JVM runtime, enabling the JIT compiler, and more.

Starting from Plug-in 1.3, the control panel is opened from the Windows Control Panel by choosing the **Java Plug-in Control Panel** item.

Java Console

The output messages sent by a Java applet to *stdout* or *stderr* are printed to the Plug-in Java Console. Enabling the Java Console is done from the Java Plug-in Control Panel in the Basic tab. When an applet is running using the Plug-in, the only console that needs to be invoked is the Plug-in Console, and not the browser console.

Java Plug-in JVM Versions

When using the Java Plug-in, you can switch between different SUN JVM versions. The selection of which JVM to run is performed in the **Advanced** tab of the Java Plug-in Control Panel. You can choose to work with any of the JREs/JDKs installed on your machine. To verify which version is currently running in your Java Plug-in, look at the first two lines in the Java Console. For example, the following shows the output of a Java Plug-in of version 1.2.2, currently running the 1.1.7B JVM:

```
Java(TM) Plug-in: Version 1.2.2.px
Using JRE version 1.1.7B
```

HTML Java Plug-in Tags

For the browser to identify a Web page that should use a Java Plug-in, certain tags must replace the <Applet> tag. Each browser suggests its own HTML tags to handle the Java Plug-in. Internet Explorer looks for an <Object> tag with the specific Java Plug-in and Applet attributes, and then uses Microsoft's COM/ActiveX technology to load it. Netscape Navigator

looks for an <Embed> tag with the specific Java Plug-in and Applet attributes, in order to use Navigator's plug-ins architecture to load the applet. You can combine both of the tags in your html page so that the applet can be loaded through the Java Plug-in using both Netscape and Internet Explorer.

There is a simple tool called the *Java Plug-in HTML Converter* that easily converts html pages to work with the Java Plug-in.

Other Environments

IBM

IBM has its own Java tools. These tools include a Java compiler, a virtual machine and slightly different JDK runtime classes.

Oracle Jinitiator

Oracle has its own virtual machine called Jinitiator. This virtual machine can be used for applications or as a plug-in for browsers. It has its own JDK classes, based on SUN's JDK 1.1.5 or JDK 1.1.7, with various changes. This virtual machine can be identified by using "*java -version*". This should result in the JDK version with an "o" at the end (e.g., "*java version 1.1.5o*"). The Jinitiator plug-in can be identified by looking at the HTML pages or by looking for the directory "*<Drive>:\Program Files\Oracle\Jinitiator <version>*".

BEA WebLogic

BEA has its own set of tools, although it does not have a virtual machine. It has its own RMI compiler (*rmic.exe*). This is used to generate stubs and skeletons, which are different than the ones produced by SUN's *rmic* tool.

VBJ, OWJAVA

Visigenic and Iona have their own *java.exe*. Usually, they come with their CORBA packages and installations. These are not newly implemented virtual machines, but rather wrappers of SUN's virtual machine with some additional functionality related to CORBA. They receive the same arguments as SUN's virtual machine.

Gemstone/J

Gemstone has its own JVM and JDK versions. If you run the `java -version` command the following messages are issued:

```
java version "3.2p2"  
HotSpot VM (3.2p2, mixed mode, build 3.2p2-Wed-Feb-23-17:58:31-PST-  
2000-build-97)
```

Frequently Asked Questions

Question 1: How do I redirect the output from an application to a file?

To redirect the *stdout* and *stderr* of a Java Application into a file, add “>out.txt 2>&1” at the end of the `java.exe` command-line. This redirects the output to a file named `out.txt`. (NT only)

Answer: In Internet Explorer, you can redirect the output by enabling the Java logging option: Right-click on the Internet Explorer icon and choose **Properties**. Click the **Advanced** tab of the **Java VM** section, and select the **Java logging enabled** check box. Restart the browser.

The output is now redirected to the file <Winnt>\Java\javalog.txt.

Question 2: How do I know whether a certain class is in the classpath?

Answer: You can use the *javap.exe* command-line tool. Run `javap.exe <full class name>` (without the “.class” suffix). When the class is not in the CLASSPATH, you will receive the following error message: ‘Class <name> not found’. If the class is in the CLASSPATH, you will receive all its fields and methods.

Question 3: How do I dump classes from the browser?

Answer: Netscape allows you to capture server classes while running an applet, and to dump them on your local machine. Typing the letter ‘L’ on the Java Console enables and disables this option. You will see the message: ‘# Class file capture enabled’ in the Java Console, and all of the server classes

will be dumped into *<Netscape folder>\Communicator\Programs\<directory name constructed from the URL address>*.

Question 4: I don't see an option in my browser to open a Java console. Where is it?

Answer: In Internet Explorer, if you don't see the option under the [View] menu, you need to enable this option in the Internet Explorer configuration. To enable the option, right-click the Internet Explorer icon and select the **Properties** option. Then, in the 'Advanced' tab, in the 'Java VM' section, select the 'Java console enabled' check box. Restart the browser. In Netscape, the option is usually placed under the [Communicator > Tools] menu, or, in earlier versions, directly under the [Tools] or [Communicator] menu.

Question 5: Why don't I see any printouts in the browser's Java Console when running with the Java Plug-in?

Answer: The browser's Java Console only shows messages sent from the browser's VM. When using the plug-in, you should work with the Java Console of the Java Plug-in.

Question 6: How can I open the Java Console when I use a Java Plug-in?

Answer: In the Plug-in's control panel, select **Start Menu > Programs > The Java Plug-in Control Panel**. In the 'Basic' tab, select the 'Show Java Console' check box.

Question 7: I have both JDK1.1 and JDK 1.2 installed on my machine. The "PATH" points to *<JDK1.1>\bin*. Why do I still see JDK1.2 when I run "*java -version*"? How can I negate JDK1.2 without uninstalling it?

Answer: During the installation of JDK1.2.x, its java.exe is also placed under the *<Winnt>\system32* directory. Because this directory is also in the PATH, the java.exe of JDK1.2 is being used. You can negate JDK1.2 without uninstalling it by renaming/removing the java.exe under the *\system32* directory.

Question 8: I get an error with *stack-trace*. How do I know if the JIT is operative?

Answer: The *stack-trace* contains the stack of invocations that lead to an error. Each line contains the name of the class and the method involved. The line number of the invocation should appear in parenthesis. If the parenthesis contains “Compiled Code” instead, this means that the bytecode was compiled to native code and the JIT is operative.

Question 9: How do I disable the JIT in java.exe?

Answer: Use the java.exe ‘-nojit’ command-line option. In JDK1.2.x and JDK1.3, there is no ‘-nojit’ option. In these versions, use the java.exe command line with the following argument:

```
-Djava.compiler=NONE
```

Question 10: How do I disable the JIT in browsers?

Answer: The solution depends on your browser:

Internet Explorer – Right-click the Internet Explorer icon and choose the **Properties** option. In the **Advanced** tab, in the **Java VM** section, select the ‘JIT compiler for virtual machine enabled’ check box. Restart the browser.

Netscape – Rename the DLL that is responsible for the JIT, so that it will not be found. The full path to the DLL is: <Netscape folder>\Communicator\Programs\java\bin\jit3240.dll

Question 11: Why do I get “NoClassDefFoundError”?

“Can't find class java.lang.NoClassDefFoundError. (Wrong class path?)”

“Exception in thread "main" java.lang.NoClassDefFoundError: Files”

Answer: These error messages are most likely to appear because of wrong usage of the *-Xbootclasspath* argument of JDK1.2. Check whether you specified the full path to the rt.jar file, located under <JDK>\jre\lib\rt.jar. Check for spaces in your parameter. If you have spaces, you must use quotes as follows:

```
-Xbootclasspath:"<...>"
```

Question 12: Why do I get “Unable to initialize threads” Error?

Answer: If you are using JDK1.1.x, check that you have `<JDK>\lib\classes.zip` in the CLASSPATH environment variable. If you are using JDK1.2.x and the `-Xbootclasspath` option, check that you specified the full path for the `rt.jar` file, located in `<JDK>\jre\lib\rt.jar`.

Question 13: Why do I get “UnsatisfiedLinkError”?

This error occurs if a Java application uses native methods of a certain library (or DLL) and the library file cannot be found, is corrupted, or is inaccessible from a specific class due to Java security restrictions.

Question 14: How can I solve “OutOfMemoryError”?

Answer: The `OutOfMemoryError` can occur if the virtual machine runs out of stack or heap size during application execution. Most virtual machines initiate their heap and stack size between 16M to 64M. Some applications may need more memory due to heavy memory usage or long recursive calls. Applications that load a large number of classes (or jars) may also consume memory at startup. In order to solve the problem, run the virtual machine with “-ms” and “-mx” arguments (“-Xms”, “-Xmx” in JDK 1.2). Note that the maximum stack and heap size is 512M (virtual memory).

Note: This section mentioned several issues that may shortly become obsolete. Although we do attempt to keep the information current, we would appreciate all feedback regarding this document at documentation@merc-int.com.

B

EJB Architecture and Testing

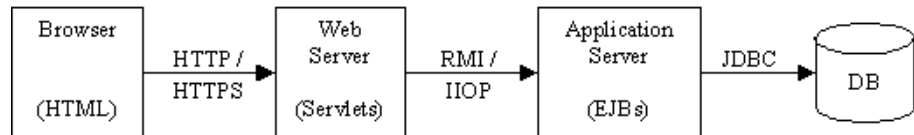
What are EJBs?

Enterprise Java Beans (EJB) are a component architecture for the development and deployment of component-based distributed business applications.

EJB systems provide complex middleware enterprise features allowing the developers to focus on the actual business architecture of the model and quickly create scalable, transactional and multi-user secure enterprise applications.

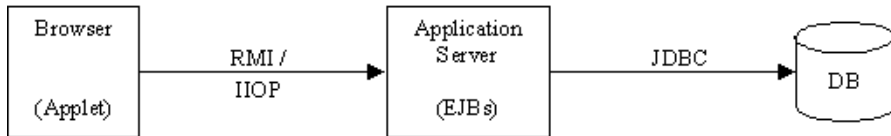
EJB Architecture

Many IT systems today are built using a well-defined 4-tier architecture or n-tier architecture. These architectures generally contain some components based on the Java technology. Notice in the diagram below how the Web Server uses Servlets and the Application Server uses EJBs. The EJBs themselves use either Entity EJBs wrapping additional tiers or directly using JDBC to talk to the database.



Some systems implement a 3-tier architecture where the Web Server and Application Server run on the same host. Although these two functions run on the same host, they still function using Servlets and EJBs. Although less

common, there are also “pure” 3-tier architecture where the Browser loads a Java applet which talks directly to EJBs on the application server, using RMI, IIOP or RMI over IIOP Protocols.



EJB Structure and Mechanism

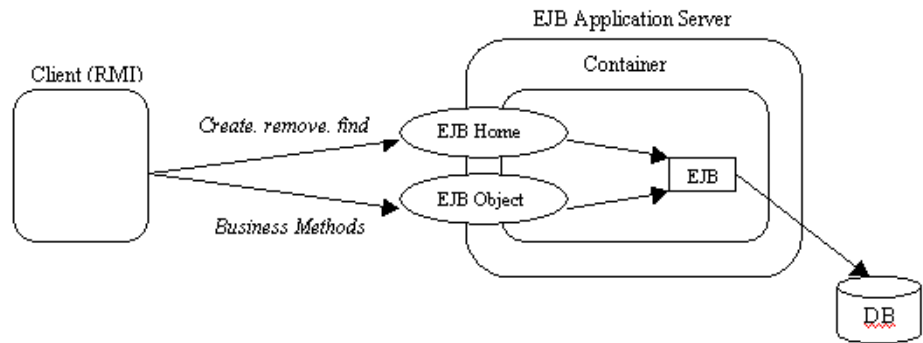
A simple EJB is constructed from 3 basic Java objects (Java classes). These classes are created by an EJB Developer and describe the EJB and its functionality.

The Java objects are: Home interface, Remote interface and Bean implementation.

- ▶ **Home interface:** Provides life cycle services for the EJB, using methods like: create(), remove(), find().
- ▶ **Remote interface:** Contains the business operations of the EJB.
- ▶ **Bean Implementation:** Implements the home and remote interface methods.

The developer can use these objects combined with a deployment descriptor file and the EJB compiler (ejbc) to generate the stubs and skeletons required to remotely access the EJB's functions called business methods. The EJB Compiler stores the entire bean in an archive file called a JAR or EAR file, similar to the way WinZip stores files and directories. The generated stubs and skeletons wrap all the lower level communication away from the programmer so the EJB object does not need to worry about the details involved with client/server communication. When deploying the EJB to the application server, the stub and skeleton classes are used by the server's EJB container to connect between the EJB and clients or other EJBs addressing it. This container also provides classes for EJB services such as: transaction, security, etc.

When the application server starts, it adds or deploys a named reference to the home stub into a JNDI tree for each EJB that is deployed in the system. JNDI or Java Naming Directory Interface is an extension to the Java language that provides an abstract way to reference resources by name in a tree structure similar to the file structure of a hard drive. These home stubs are then available through JNDI for use by clients of the application server. Each client downloads a copy of the home stub for use by the client locally on its machine. The client uses the Home stub to create an EJB instance on the application server. On the application server, the home skeleton (controlled by the container) creates an EJB instance, gives it a state, associates a client context, creates a remote skeleton, and creates a remote stub. The home skeleton returns the remote stub to the client. Now the client can use this remote stub locally to invoke business methods on the EJB.



Deployable EJB JAR should contain the Home interface, the Remote interface, the Bean implementation, the stub and skeleton classes, the Deployment Descriptor files and any referenced Java class files not part of the container's classpath. (The classpath is an environment variable that Java uses to locate class files and archives containing class files). The Deployment Descriptor files contain information such as the EJB name, class, home and remote interfaces, bean type (session or entity), environment entries, resource factory references, EJB references, and the JNDI name.

There are 2 kinds of Deployment Descriptors:

- **Serialized Deployment Descriptor:** A file that contains serialized information on an EJB. The file has ".ser" extension. [EJB1.0 specification]

- **XML Deployment Descriptor:** XML file that contains information about all EJB objects in the JAR file. The name of this file is `ejb-jar.xml`. These descriptors are stored in the `META-INF` directory in the jar or ear file. Some EJB containers have additional deployment descriptor files needed for the EJB deployment. For example, Weblogic, requires `weblogic-ejb-jar.xml`. [EJB1.1 specification]

There are 3 types of EJB objects:

- **Stateless Session Bean:** Provides a single-use service and does not maintain any state across multiple calls on behalf of the client.
- **Stateful Session Bean:** Provides conversational interaction and maintains a state across multiple calls.
- **Entity Bean:** A persistent EJB, generally used to represent data in a database.

Deployment in Common Application Servers

Type	EJB Specification	Deployment Descriptor	How to Deploy?
WebLogic4.x	1.0	.ser files	- Create Deployable Jar using ejbc tool.
WebLogic5.x	1.1	ejb-jar.xml weblogic-ejb-jar.xml	- Deploy by specifying full path to EJB JAR file in the <WL>/weblogic.properties Property: weblogic.ejb.deploy
WebLogic6.x	1.1, 2.0	ejb-jar.xml weblogic-ejb-jar.xml	- Create Deployable Jar using ejbc tool. - Deploy by copying the EJB JAR/EAR to directory: <WL>/config/<domain>/applications
WebSphere3.0	1.0	.ser files	- Create Deployable Jar using jetace tool.
WebSphere3.5	1.0, 1.1 partially		- Deploy by the administrative console UI. (Deployed EJBs will be placed in directory: <WS>/deployedEJB)
WebSphere 4.0	1.0, 1.1	ejb-jar.xml	- Use Application Assembly tool to create bean and deploy
Oracle OC4J	1.1, 2.0 partially	ejb-jar.xml orion-ejb-jar.xml	-Create .ear file or, alternatively, a directory built according to .ear file structure. -Register your .ear/directory within server.xml configuration file or place it under applications directory in OC4J installation.
Sun J2EE	1.1	ejb-jar.xml	-Create.ear file and use the application deployment tool to deploy the application.

EJB Unit-Testing

Until now, testing EJBs was done as part of an RMI-Vuser, by recording the EJB Home and Remote interfaces API driven by walking through the GUI. Such testing was not possible in the absence of a client implementation. In such cases, Java Vuser was used to manually create an EJB testing script. The latter solution is complex and requires full understanding of and access to source code.

It is reasonable to assume that some customers will build their system from the application server and up. This means that they will implement the application server side first and check its scalability, and only then implement the back tiers (i.e. Web-server/client side). ProTune can be used in the early development phase to functionally test the methods and interfaces of an EJB using the parameterization abilities and java functionality of VuGen alone, since the lack of a user interface prevents the use of traditional functional testing tools such as WinRunner. Additionally, ProTune will assist with testing the scalability issue by combining EJB Vusers with the Console and Analysis tools.

The EJB Unit-Testing solution allows you to browse through EJBs installed on an application server, and automatically generate full and ready-to-run script for a selected EJB.

C

Calling External Functions

When working with VuGen, you can call functions that are defined in external DLLs. By calling external functions from your script, you can reduce the memory footprint of your script and the overall run-time.

To call the external function, you load the DLL in which the function is defined.

You can load a DLL:

- ▶ locally—for one script, using the **lr_load_dll** function
- ▶ globally—for all scripts, by adding statements to the `vugen.dat` file

Loading a DLL—Locally

You use the **lr_load_dll** function to load the DLL in your Vuser script. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

To call a function defined in a DLL:

- 1** Use the **lr_load_dll** function to load the DLL at the beginning of your script. Place the statement at the beginning of the *vuser_init* section. **lr_load_dll** replaces the **ci_load_dll** function.

Use the following syntax:

```
lr_load_dll(library_name);
```

Note that for UNIX platforms, DLLs are known as shared libraries. The extension of the libraries is platform dependent.

- 2** Call the function defined in the DLL in the appropriate place within your script.

In the following example, the *insert_vals* function, defined in *orac1.dll*, is called, after the creation of the Test_1 table.

```
int LR_FUNC Actions(LR_PARAM p)
{
    lr_load_dll("orac1.dll");

    lrd_stmt(Csr1, "create table Test_1 (name char(15), id integer)\n", -1,
              1 /*Deferred*/, 1 /*Dflt Ora Ver*/, 0);
    lrd_exec(Csr1, 0, 0, 0, 0, 0);

    /* Call the insert_vals function to insert values into the table. */
    insert_vals();

    lrd_stmt(Csr1, "select * from Test_1\n", -1, 1 /*Deferred*/, 1 /*Dflt Ora
    Ver*/, 0);
    lrd_bind_col(Csr1, 1, &NAME_D11, 0, 0);
    lrd_bind_col(Csr1, 2, &ID_D12, 0, 0);
    lrd_exec(Csr1, 0, 0, 0, 0, 0);
    lrd_fetch(Csr1, -4, 15, 0, PrintRow14, 0);
    ...
}
```

Note: You can specify a full path for the DLL. If you do not specify a path, **lr_load_library** searches for the DLL using the standard sequence used by the C++ function, `LoadLibrary` on Windows platforms. On UNIX platforms you can set the `LD_LIBRARY_PATH` environment variable (or the platform equivalent). The `lr_load_dll` function uses the same search rules as *dlopen*. For more information, see the man pages for *dlopen* or its equivalent.

Loading a DLL—Globally

You can load a DLL globally, to make its functions available to all your Vuser scripts. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

To call a function defined in a DLL:

- 1 Add a list of the DLLs you want to load to the appropriate section of the *mdrv.dat* file, located in the VuGen/dat directory.

Use the following syntax,

```
PLATFORM_DLLS=my_dll1.dll, my_dll2.dll, ...
```

replacing the word *PLATFORM* with your specific platform. For a list of platforms, see the beginning section of the *mdrv.dat* file.

For example, to load DLLs for Winsocket Vusers on an NT platform, add the following statement to the *mdrv.dat* file:

```
[WinSock]
ExtPriorityType=protocol
WINNT_EXT_LIBS=wrun32.dll
WIN95_EXT_LIBS=wrun32.dll
LINUX_EXT_LIBS=liblrs.so
SOLARIS_EXT_LIBS=liblrs.so
HPUX_EXT_LIBS=liblrs.sl
AIX_EXT_LIBS=liblrs.so
LibCfgFunc=winsock_exten_conf
UtilityExt=lrun_api
ExtMessageQueue=0
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
WINNT_DLLS=user_dll1.dll, user_dll2.dll, ...
```

- 2 Call the function defined in the DLL in the appropriate place within your script.

D

Programming Scripts on UNIX Platforms

ProTune users on UNIX platforms can create Vuser scripts through programming. To create a script through programming, you use a ProTune template.

This appendix describes:

- ▶ Generating Templates
- ▶ Programming Vuser Actions into a Script
- ▶ Configuring Vuser Run-Time Settings
- ▶ Defining Transactions and Rendezvous Points
- ▶ Compiling Scripts

About Programming Vuser Scripts to Run on UNIX Platforms

There are two ways to create Vuser scripts that run on UNIX platforms: by using VuGen, or by programming.

VuGen You can use VuGen to create Vuser scripts that run on UNIX platforms. You record your application in a Windows environment and run it in UNIX—recording is not supported on UNIX.

programming Users working in UNIX-only environments can program Vuser scripts. Scripts can be programmed in C or C++ and they must be compiled into a dynamic library.

This appendix describes how to develop a Vuser script by programming.

To create a script through programming, you can use a ProTune template as a basis for a larger Vuser script. The template provides:

- correct program structure
- ProTune API calls
- source code and makefiles for creating a dynamic library

After creating a basic script from a template, you can enhance the script to provide run-time Vuser information and statistics. For more information, see Chapter 6, “Enhancing Vuser Scripts.”

Generating Templates

ProTune includes a utility that copies a template into your working directory. The utility is called *mkdbtest*, and is located in `$M_LROOT/bin`. You run the utility by typing:

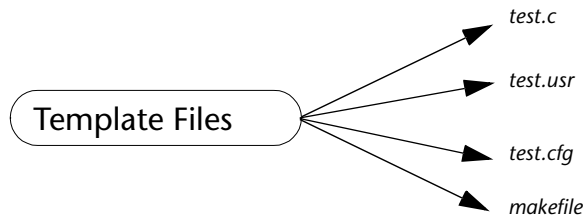
```
mkdbtest name
```

When you run *mkdbtest*, it creates a directory called *name*, which contains the template file, *name.c*. For example, if you type

```
mkdbtest test1
```

mkdbtest creates a directory called *test1*, which contains the template script, *test1.c*.

When you run the *mktbtest* utility, a directory is created containing four files *test.c*, *test.usr*, *test.cfg* and *Makefile*, where *test* is the test name you specified for *mktbtest*.



Programming Vuser Actions into a Script

The Vuser script files, *test.c*, *test.usr*, and *test.cfg*, can be customized for your Vuser.

You program the actual Vuser actions into the *test.c* file. This file has the required structure for a programmed Vuser script. The Vuser script contains three sections: *vuser_init*, *Actions*, and *vuser_end*.

Note that the template defines *extern C* for users of C++. This definition is required for all C++ users, to ensure that none of the exported functions are inadvertently modified.

```
#include "lrun.h"
#if defined(__cplusplus) || defined(cplusplus) extern "C"
{
#endif
int LR_FUNC vuser_init(LR_PARAM p)
{
    lr_message("vuser_init done\n");
    return 0;
}
int Actions(LR_PARAM p)
{
    lr_message("Actions done\n");
    return 0;
}
int vuser_end(LR_PARAM p)
{
    lr_message("vuser_end done\n");
    return 0;
}
#endif
#endif
```

You program Vuser actions directly into the empty script, before the **lr_message** function of each section.

The *vuser_init* section is executed first, during initialization. In this section, include the connection information and the logon procedure. The *vuser_init* section is only performed once each time you run the script.

The *Actions* section is executed after the initialization. In this section, include the actual operations performed by the Vuser. You can set up the Vuser to repeat the Actions section (in the *test.cfg* file).

The *vuser_end* section is executed last, after the all of the Vuser's actions. In this section, include the clean-up and logoff procedures. The *vuser_end* section is only performed once each time you run the script.

Note: ProTune controls the Vuser by sending SIGHUP, SIGUSR1, and SIGUSR2 UNIX signals. Do not use these signals in your Vuser programs.

Configuring Vuser Run-Time Settings

To configure Vuser run-time settings, you modify the *default.cfg* and *default.usp* files created with the script. These run-time settings correspond to VuGen's run-time settings. (See "Configuring Run-Time Settings" on page 115.) The *default.cfg* file contains the setting for the General, Think Time, and Log options. The *default.usp* file contains the setting for the Run Logic and Pacing.

General Options

There is one General options for Unix Vuser scripts:

ContinueOnError instructs the Vuser to continue when an error occurs. To activate the option, specify 1. To disable the option, specify 0.

In the following example, the Vuser will continue on an error.

```
[General]
ContinueOnError=1
```

Think Time Options

You can set the think time options to control how the Vuser uses think time during script execution. You set the parameters Options, Factor, LimitFlag, and Limit parameters according to the following chart.

Option	Options	Factor	LimitFlag	Limit
Ignore think time	NOTHINK	N/A	N/A	N/A
Use recorded think time	RECORDED	1.000	N/A	N/A
Multiply the recorded think time by...	MULTIPLY	number	N/A	N/A
Use random percentage of recorded think time	RANDOM	range	lowest percentage	upper percentage
Limit the recorded think time to...	RECORDED / MULTIPLY	number (for MULTIPLY)	1	value in seconds

To limit the think time used during execution, set the LimitFlag variable to 1 and specify the think time Limit, in seconds.

In the following example, the settings tell the Vuser to multiply the recorded think time by a random percentage, ranging from 50 to 150.

```
[ThinkTime]
Options=RANDOM
Factor=1
LimitFlag=0
Limit=0
ThinkTimeRandomLow=50
ThinkTimeRandomHigh=150
```


Log Options

You can set the log options to create a brief or detailed log file for the script's execution.

```
[Log]
LogOptions=LogBrief
MsgClassData=0
MsgClassParameters=0
MsgClassFull=0
```

You set the parameters `LogOptions`, `MsgClassData`, `MsgClassParameters`, and `MsgClassFull` variables according to the following chart.

Logging Type	LogOptions	MsgClassData	MsgClassParameters	MsgClassFull
Disable Logging	LogDisabled	N/A	N/A	N/A
Standard Log	LogBrief	N/A	N/A	N/A
Parameter Substitution (only)	LogExtended	0	1	0
Data Returned by Server (only)	LogExtended	1	0	0
Advanced Trace (only)	LogExtended	0	0	1
All	LogExtended	1	1	1

In the following example, the settings tell the Vuser to log all data returned by the server and the parameters used for substitution.

```
[Log]
LogOptions=LogExtended
MsgClassData=1
MsgClassParameters=1
MsgClassFull=0
```

Iterations and Run Logic

You can set the Iteration options to perform multiple iterations and control the pacing between the iterations. You can also manually set the order of the actions and their weight. To modify the run logic and iteration properties of a script, you must edit the *default.usp* file.

To instruct the Vuser to perform multiple iterations of the Actions section, set `RunLogicNumOfIterations` to the appropriate value.

To control the pacing between the iterations, set the `RunLogicPaceType` variable and its related values, according to the following chart.

Pacing	RunLogicPaceType	Related Variables
As soon as possible	Asap	N/A
Wait between Iterations for a set time	Const	RunLogicPaceConstTime
Wait between iterations a random time	Random	RunLogicRandomPaceMin, RunLogicRandomPaceMax
Wait after each iteration a set time	ConstAfter	RunLogicPaceConstAfterTime
Wait after each iteration a random time	After	RunLogicAfterPaceMin, RunLogicAfterPaceMax

In the following example, the settings tell the Vuser to perform four iterations, while waiting a random number of seconds between iterations. The range of the random number is from 60 to 90 seconds

```
[RunLogicRunRoot]
MerClniTreeFather=""
MerClniTreeSectionName="RunLogicRunRoot"
RunLogicRunMode="Random"
RunLogicActionOrder="Action,Action2,Action3"
RunLogicPaceType="Random"
RunLogicRandomPaceMax="90.000"
RunLogicPaceConstTime="40.000"
RunLogicObjectKind="Group"
RunLogicAfterPaceMin="50.000"
Name="Run"
RunLogicNumOfIterations="4"
RunLogicActionType="VuserRun"
RunLogicAfterPaceMax="70.000"
RunLogicRandomPaceMin="60.000"
MerClniTreeSons="Action,Action2,Action3"
RunLogicPaceConstAfterTime="30.000"
```

Defining Transactions and Rendezvous Points

When programming a Vuser script without VuGen, you must manually configure the Vuser file in order to enable transactions and rendezvous. The configuration settings are listed in the *test.usr* file.

```
[General]
Type=any
DefaultCfg=Test.cfg
BinVuser=libtest.libsuffix
RunType=Binary

[Actions]
vuser_init=
Actions=
vuser_end=

[Transactions]
transaction1=

[Rendezvous]
Meeting=
```

Each transaction and rendezvous must be defined in the *usr* file. Add the transaction name to the Transactions section (followed by an “=”). Add each rendezvous name to the Rendezvous section (followed by an “=”). If the sections are not present, add them to the *usr* file as shown above.

Compiling Scripts

After you modify the template, you compile it with the appropriate *Makefile* in the script's directory. Note that for C++ compiling, you must use the native compiler (not gnu). The compiler creates a dynamic library called:

- libtest.so (solaris)
- libtest.a (AIX)
- libtest.sl (HP)

You can modify the *Makefile* and assign additional compiler flags and libraries by modifying the appropriate sections.

If you are working with a general template, you must include your application's libraries and header files. For example, if your application uses a library called *testlib*, include it in the LIBS section.

```
LIBS      = \  
          -testlib \  
          -lrun50 \  
          -lm
```

After you modify the *makefile*, type Make from the command line in the working directory to create the dynamic library files for the Vuser script.

You can now run the script from the ProTune Console. The Vuser script is the *script.usr* file located in the script's directory. For information on how to integrate a Vuser script into a session step, refer to the appropriate *ProTune Console User's Guide*.

Before integrating a script into a session step, you should run it from the command line to verify that it works properly.

To run a Vuser script from the UNIX command line, type:

```
mdrv -usr 'pwd' test.usr
```

where *pwd* is the full path to the directory containing the Vuser script and *test.usr* is the name of the Vuser file. Check that your script communicates with the server and performs all the required tasks.

E

Using Keyboard Shortcuts

The following list describes the keyboard shortcuts available in the Virtual User Generator.

ALT+F8	Compares the Current Snapshots (Web Vusers only)
ALT+INS	Create New Step
CTRL+A	Select All
CTRL+C	Copy
CTRL+F	Find
CTRL+G	Go To Line
CTRL+H	Replace
CTRL+N	New
CTRL+O	Open
CTRL+P	Print
CTRL+S	Save
CTRL+V	Paste
CTRL+X	Cut
CTRL+Y	Redo
CTRL+Z	Undo
CTRL+F7	Recording Options
CTRL+F8	Scan for Correlations

CTRL+SHIFT+SPACE	Show Function Syntax (Intellisense)
CTRL+SPACE	Complete Wizard (completes the function name)
F1	Help
F3	FIND Next Downward
SHIFT+F3	Find Next Upward
F4	Run-Time Settings
F5	Run Vuser
F6	Move Between Panes
F7	Show EBCDIC Translation Dialog (for WinSocket data)
F9	Toggle Breakpoint
F10	Run Vuser Step by Step

Index

A

- ABC icon 78
- Acrobat Reader xix
- Action
 - method 316
 - section 26
- Action steps
 - function list-Web 406
 - modifying - Web 496
- actions
 - importing 39
 - recording multiple 32
 - reordering 39
- Actions class 315
- Add New Column dialog box 95
- Additional Attributes run-time settings 131
- advanced copy and paste (WinSock) 284
- Advanced Correlation Properties dialog box 525
- Advanced recording options 435
- animated run
 - defined 140
 - enabling 140
- ANSI C support
 - for enhancing scripts 73
- application server, Oracle NCA 600
- automatic proxy configuration script 460
- automatic transactions
 - Database Vuser scripts 235
 - general 137
- automation compliant 306
- autorecovery 13

B

- Baan Vuser scripts
 - customizing 630

- Baan Vuser scripts (*cont'd*)
 - function list 624
 - overview 623
- Bearer settings (WAP) 755
- bearers support (WAP) 741
- binary coded data 494
- binary view of data (WinSock) 278
- bookmarks in data (WinSock) 282
- Books Online xix
- boundaries
 - defining for correlation (Web) 544
- braces, using in parameterization 105
- breakpoints 143
- Brief log run-time setting 128
- browser
 - launching (Web/WinSock) 421
 - manual launching (Web) 444
 - recording options (Web) 444
 - specifying location (Web) 444
 - using the default (Web) 444
- browser cache (Web, Wireless) 464
- Browser Emulation settings, Web 462
- buffer navigator (WinSock) 280
- buffer size on network (Internet) 471

C

- C functions
 - additional keywords 784
 - calling libc functions 74
 - for debugging 783
 - limitations in Vuser scripts 74
 - using in Vuser scripts 16
- C language support
 - conventions 307
 - interpreter 17
- C Vusers 307

Creating Vuser Scripts

- cache, browser (Web, Wireless) 464
- CARRAY buffers 710
- Check Properties dialog box 514
- CHECK_HRES (COM error checking) 347
- checks (Web)
 - defining additional properties 488
 - functions 407
 - image checks 485
 - modifying in scripts 514
 - overview 477
 - text 480
 - types of 479
- cHTML 728
- Citrix ICA Vuser scripts
 - recording 207
- Citrix server, disconnecting 224
- Citrix Vuser scripts
 - display settings 220
 - getting started 208
 - recording log 219
 - run-time settings 221
 - setting recording options 210
 - synchronizing replay 213
 - tips for record and replay 224
 - using ctrx functions 216
 - viewing 218
- Classpath
 - recording options 176
 - run-time settings 202
- client-side certificates 561
- Close All command 147
- code generation options (EJB) 584
- Collapse All command 556
- COM
 - data types 335
 - overview and interfaces 334
- COM Vuser scripts
 - creating object instances 348, 360
 - developing 333
 - error checking 347
 - functions 359
 - getting started 335
 - IDispatch interface 351
 - instantiating objects 348, 360
 - Irc_*type* functions 361
 - recording options 338
 - COM Vuser scripts (*cont'd*)
 - retrieving an interface 349
 - scanning for correlations 354
 - script structure 346
 - selecting COM objects to record 337
 - type libraries 334
 - understanding 346
 - Visual Basic collection 368
 - command line arguments
 - handling (C Vusers) 72
 - handling (Java Vusers) 325
 - UNIX Vuser scripts 148
 - command prompt 147
 - Comment button 64
 - comments
 - inserting into Vuser scripts 64
 - screen header comments (RTE) 651
 - comparing Vusers 111
 - comparison method 533, 572
 - compiling Vuser scripts (UNIX) 852
 - complete word 22
 - compression for HTML (gzip) 465
 - concurrent group functions 408
 - Connect dialog box (RTE) 647
 - connection attempts, modifying (RTE) 660
 - Connection to TestDirector dialog box 151
 - content type filtering (Web) 439
 - Content type filters dialog box 439
 - ContentCheck settings (Web) 473
 - Context Sensitive Help xix
 - context, resetting 436
 - Continue on error 69, 133
 - Control steps
 - modifying (Web) 510
 - converting
 - custom request to C 508
 - Web functions to Java 402
 - copy and pasting for RTE Vusers 648
 - Corba recording options. 187
 - Corba-Java Vuser scripts 371
 - correlation options 183
 - debug options 184, 187
 - Recorder options 177
 - recording 372
 - serialization options 181
 - Correlated Query tab 255

- correlating
 - after recording (Web, Wireless) 527
 - COM Vusers 354
 - for known contexts (Web) 517
 - functions (C) 109
 - functions (Java) 110
 - HTML statements (Web) 515
 - Java statements 189
 - missing correlations (Database) 799
 - modifying existing parameters 113
 - overview 107
 - recording options-Java 183
 - rules for Web Vusers 518
 - scanning Database Vuser script 254
 - scripting language options 46
 - Tuxedo 713
 - with snapshots (Web) 527
 - Correlation options
 - in Script recording options 46
 - Correlation Results tab 536
 - Correlation tab 523
 - CtLib 231
 - logging server messages 128
 - options 236
 - result set errors 251
 - ctx functions 216
 - custom headers
 - Web, Wireless 437
 - Custom Request dialog box (Web) 509
 - Custom Request step
 - defined 413
 - for XML 549
 - modifying (Web) 508
 - custom requests 453
 - custom scripts
 - C Vusers 307
 - Java Vusers 309
 - JavaScript Vusers 312
 - VB Vusers 310
 - VBScript Vusers 311
- D**
- data buffers
 - Tuxedo Vuser scripts 706
 - WinSock Vuser scripts 290
 - data files
 - used for parameterization 91
 - Windows Sockets Vuser scripts 291
 - data grids, show/hide 147
 - Data Wizard 100
 - Database Query Wizard dialog box 100
 - Database recording options 235
 - Database Vuser scripts
 - correlating 253
 - developing 231
 - getting started 234
 - handling errors 250
 - return codes 249
 - row information 247
 - tips 791
 - using lrd functions 239
 - viewing grids 246
 - date/time, parameter values 83
 - DB2-CLI 231
 - DbLib 231
 - DCOM 334
 - DCOM tab 338
 - Debug Message dialog box 68
 - Debug recording settings (Java) 184
 - debugging
 - database applications 785
 - during replay 143
 - enabling for Web Vusers 145
 - obtaining information (WAP) 471
 - Oracle applications 787
 - setting debug level 128
 - Web Vuser scripts 553
 - decrypting text 72
 - deep correlation (Java) 190
 - defining parameter properties 82
 - data files 94
 - defining properties, text checks 488
 - deleting steps
 - from Web scripts 496
 - delimiters, in data tables 96
 - detector, EJB 576
 - device name (RTE) 660
 - disable logging log option 126
 - disconnecting from TestDirector 153
 - Display tab, General options 145
 - distinguished names 395

Creating Vuser Scripts

- DLLs, calling from a Vuser script 839
- DN (LDAP) 395
- DNS caching
 - Web 469
- DNS Vusers
 - functions 264
 - overview 263
- documentation set xx
- DSL 466
- dual protocol (Web/WinSock) 417
- duplicate key violations
 - Oracle, MSSQL 793
 - Siebel 796
- dynamic ports 298

E

- EBCDIC translation 293
- editor font 13
- EJB
 - architecture 833
 - code generation options 584
 - instance 587
 - method 589
 - Vuser scripts 575
- EJB Detector 587
 - command-line 577
 - log files 580
 - setup 576
- encrypting text 72
- end method 315
- End Transaction dialog box 62
- environment settings
 - Java 326
 - Tuxedo Vusers 707
- Environment tab 13
- error handling 69, 133
 - COM Vuser scripts 347
 - modifying globally 250
 - modifying locally (severity level) 251
- Error Message dialog box 68
- escape sequence 296
- exception handling for Baan protocol 631
- Execution Log tab 141
- Execution report (Web only) 571
- Expand All command 556

- Expect property, Web checks 488
- exporting to a zip file 37
- extended log option 127
- extended result set 236
- external functions 839

F

- fetching data 247
- field demarcation characters 657
- FIELDTBL environment setting 707
- filtering
 - content type (Web, Wireless) 439
 - report information (Web) 556
- Filters dialog box (Web reports) 556
- FLDTBLDIR environment setting 707
- format
 - for parameterization 92
 - of data in display buffer 295
- Forms Listener 615
- Frame property, for object checks (Web) 488
- FTP protocol
 - function list 388
 - recording 387, 769
- full run-time trace 128
- Function Reference xix
- functions
 - Baan 624
 - ctx (Citrix) 216
 - imap 679
 - in Web Vuser scripts 405
 - Java 317
 - lr (C functions) 15
 - lrc (COM) 345
 - lrd (Database) 239
 - lreal (Real Player) 719
 - lrs (WinSock) 271
 - lrt (Tuxedo) 699
 - mapi 681
 - pop3 683
 - smtp 684
 - te (RTE) 638
 - WAP 737

G

- Gateway settings (WAP) 752
- General options
 - all Vusers 106
 - Citrix display 220
 - Display tab (Web only) 145
 - Environment tab 13
 - Parameterization tab 105
 - Replay tab 141
- Global Unique Identifier (GUID) 334
- graphs
 - enabling for Web 467
- grids
 - hiding 147
 - viewing 246
- group name, parameter values 85
- GUID 334
- gzip 465

H

- headers
 - custom 437
 - risky 437
- HotSync 427
- HTML
 - maximum parameter length 545
- HTML view (Web snapshots) 530
- HTML-based mode 445
- HTTP
 - buffer size (Web) 471
- HTTP recording mode, WAP 752
- hypertext link step
 - defined 413
 - modifying 499

I

- ica files 223
- IDispatch interface 351, 361
- If-Modified-Since header
 - Web 464
- IIOP 380
- image checks
 - modifying (Web) 500
 - Web Vuser scripts 485

- Image Step Properties dialog box 501
- IMAP protocol 677
- i-mode
 - overview 728
 - toolkits 728
- importing
 - actions 39
 - data from a database 99
- Informix 231
- init method 315
- Insert Comment dialog box 64
- Instantiating COM objects 348
- intellisense 22
- internal data, parameterization 83
- Internet Messaging (IMAP) 677
- ISDN 466
- iteration number, parameter values 86
- iterations
 - run-time settings 122
 - updating parameters for each 93
- IUnknown interface 335

J

- Jacada Vuser scripts 689
 - recording 691
 - replaying 693
 - understanding 694
- Java plug-in 377
- Java virtual machine
 - recording options 174
 - run-time settings 204
- Java Vusers (all)
 - correlating statements 189
 - editing Java methods 315
 - environment settings 326
 - inserting rendezvous points 321
 - programming 313
- Java Vusers (Corba, RMI)
 - Classpath run-time settings 202
 - Java VM run-time settings 204
 - recording options, correlation 183
 - recording options, Java VM 174, 176
 - recording options, serialization 181
 - recording tip 377

Creating Vuser Scripts

- Java Vusers (custom)
 - creating template 314
 - using Java code 309

- JavaScript Vusers 312

- JNDI properties
 - advanced, context factory 583
 - locating EJB home 586
 - specifying 582

- Jscript 44

K

- keep-alive connections, Web 470

- keyboard mapping (RTE) 640

- keyboard shortcut
 - recording options 46
 - run-time settings 116
 - shortcuts list 855

- keywords, adding additional 784

L

- language for script generation 43

- LDAP protocol
 - function list 392
 - recording 391
 - via WinSock 265

- libc functions, calling 74, 308

- libraries, for scripting 138

- Link Step Properties dialog box 499

- load balancing, Oracle NCA 617

- load generator name, parameter values 87

- loading DLLs
 - globally 841
 - locally 839
 - overview 839

- log
 - full trace recording (Web) 436
 - setting detail level - PC 127
 - setting detail level - UNIX 849

- log cache size 126

- Log Message dialog box 66, 67

- Log run-time settings 125

- lrbat utility 768

- lrc functions 345

- lrd (Database) functions 239

- lreal functions 719

- lrs functions 271

- lrt functions 699

M

- Mailing Services protocols

 - IMAP 679

 - MAPI 681

 - POP3 683

 - recording 678

 - SMTP 684

- MAPI functions 681

- mapping keyboard 640

- MatchCase property 488

- maximum length of HTML parameters 545

- Media Player 720

- messages

 - sending to output 66

- methods, Java 315

- Miscellaneous run-time settings 132

- mkdbtimpl script (UNIX) 844

- MMS functions (MS Media Player) 720

- MMSC 745

- modifying Web scripts

 - image steps 500

 - rendezvous points 512

 - submit data steps 505

 - submit form steps 502

 - think time 512

 - URL steps 497

- MS

 - Exchange protocol (MAPI) 681

 - SQL Server, recording on 231

- MTS components 341

- multi action 28

- multi protocol 28

- multithreading 136

N

- navigating through WinSock data 280

- NCA Vusers, see Oracle NCA

- network settings 466

- Network Speed, run-time setting 466

- New button 644

New Virtual User dialog box, RTE 644
 Nokia toolkits 738
 non-printable characters 296
 non-resources 440
 non-standard HTTP applications 569

O

ODBC recording 231
 offset of data in buffer (WinSock) 293
 OnFailure property, Web checks 488
 online browser 144, 567
 Open command 558
 options
 CtLib 236
 lrd log 236
 parameterization 105
 recording (RTE) 650
 Oracle 231
 Oracle application debugging 787
 Oracle Configurator 615
 Oracle NCA Vuser scripts
 creating 597
 recording guidelines 600
 run-time settings 611
 secure applications 614
 using Vuser functions 605
 Oracle version 8.0 243
 OTA, Over-The-Air 743
 Output Message dialog box 68
 Output window 322
 hiding 142
 show/hide 147

P

Pacing settings 122
 page view (Web snapshots) 530
 Palm
 protocol 417
 recording applications 427
 PAP, Push Access Protocol 743
 Parameter Properties dialog box 82
 parameter types
 data files 91
 date/time 83

parameter types (*cont'd*)
 group name 85
 internal data 83
 iteration number 86
 list 83
 load generator name 87
 Vuser ID 91
 parameterization
 brace style 105
 creating a new parameter 78
 data files 91
 defining properties 82
 internal data type formats 92
 Java 80
 modifying existing parameters 113
 naming a parameter 79
 options 105
 overview 76
 parameter list 104
 random sequence with seed 98
 restoring original value 81
 selecting a parameter type 79
 setting properties for data files 94
 Tuxedo scripts 705
 understanding parameter types 83
 undoing (Web) 82
 updating parameter values 93
 updating values from files 97
 updating with unique values 98
 using internal data 83
 pausing a Vuser 142
 PeopleSoft8 397, 403
 PeopleSoft-Tuxedo Vusers 697
 persistent connections, Web 470
 phone, recording over 740
 POP3 (Post Office) protocol 683
 Port Mapping settings 49
 PPG, Push Proxy Gateway 743
 Pragma mode 613
 Preferences run-time settings 467
 Print dialog box (Web reports) 559
 printing Results Summary reports 559
 programming
 in Visual Studio 763
 using templates 764, 844
 Vuser actions 845

Creating Vuser Scripts

properties

- Expect (Web) 488
- Frame (Web) 488
- MatchCase (Web) 488
- OnFailure (Web) 488
- Repeat (Web) 489
- Report (Web) 489
- text checks 488

properties of parameters

- defining 82
- defining for data files 94

protocols, *See* Vusers

Proxy Authentication dialog box 434

proxy server

- recording options (Web) 432
- recording options (Web/WinSock) 421
- run-time settings (Internet) 459

push support 742

R

Radius run-time settings (WAP) 758

RADIUS support 742

random parameter assignment 98

RealPlayer 717

Record button 31

recording log generation 45

Recording Log tab 38

recording mode

- i-mode, VoiceXML 749
- WAP 748

recording options

- Advanced (Web, Wireless) 435
- Browser (Web) 444
- Corba Options node 187
- Database node 235
- Debug node (Java) 184
- Internet protocols 431
- keyboard shortcut 46
- Port Mapping node 49
- Recorder node (Java) 177
- Recording (Web) 453
- Recording Proxy (Web, Wireless) 432
- Recording Proxy (Web/WinSock) 421
- RTE 650

recording options (*cont'd*)

- Script (FTP, COM, Mail) 44
- setting for Web 443
- WAP Toolkit 750
- Web 443
- WinSock node 268
- Wireless 747

recording Vuser scripts

- Baan 624
- Citrix ICA 207
- Corba-Java 372
- database 234
- DNS 263
- FTP 387
- LDAP 391
- Mailing services 677
- Oracle NCA 599
- proxy setting 432
- Rmi-Java 379
- Tuxedo 697
- Web/WinSock 417
- Window Sockets 265
- Wireless 733
- with VuGen 25

regenerating Vusers

- all protocols 40

regular expressions

- in text checks 489

rendezvous

- Rendezvous dialog box 63

rendezvous points

- inserting 63
- Java Vusers 321
- modifying in Web scripts 512

Repeat property, Web Vusers 489

Replace More Occurrences command 80

Replay tab, General Options dialog box 141

Report property, Web checks 489

Report toolbar 555

report tree, Results Summary (Web) 555

reset context 436

resources, excluding 440

restoring original value of parameter 81

Results Summary report 553

- debugging Web scripts 553
- filtering information 556

- Results Summary report (*cont'd*)
 - opening 558
 - printing 559
 - searching 557
 - tree branches 555
 - understanding 555
 - return codes 249
 - RMI-Java Vuser scripts
 - correlation options 183
 - debug options 184, 187
 - Recorder options 177
 - recording 380
 - recording over IIOP 380
 - serialization options 181
 - row count, obtaining 793
 - row information, Database Vusers 247
 - RTE Vuser scripts
 - getting started 637
 - introducing 636
 - mapping PC keyboard 640
 - overview 635
 - reading text from screen 673
 - recording 643
 - run-time settings 659
 - steps in creating 637
 - synchronizing 663
 - using te functions 638
 - Run command 141
 - Run Logic settings 117
 - run_db_vuser shell script 148
 - running Vuser scripts
 - animated mode 140
 - in stand-alone mode 139
 - using VuGen 139
 - run-time settings
 - Additional Attributes 131
 - all protocols 115
 - Bearers node (WAP) 755
 - Browser Emulation node 462
 - Client Emulation (Oracle NCA) 613
 - configuring manually 847
 - ContentCheck node (Web) 473
 - debug information (WAP) 471
 - dialog box 116
 - Gateway node (WAP) 752
 - Internet protocols (Web etc.) 457
 - run-time settings (*cont'd*)
 - keyboard shortcut 116
 - Log node 125
 - Miscellaneous 132
 - Oracle NCA 611
 - Pacing node 122
 - Preferences - Advanced 468
 - Preferences node (Internet prot.) 467
 - Proxy node (Internet prot.) 459
 - Radius node (WAP) 758
 - RTE 666
 - RTE node 659
 - Run Logic node 117
 - shortcuts 116
 - Speed Simulation (Internet prot.) 466
 - Speed Simulation (Oracle NCA) 611
 - Think Time 129
 - VBA node 137
 - WAP 751
 - run-time viewer
 - enabling in VuGen 144
 - tips for using (VuGen) 567
- S**
- S_SSA_ID table 799
 - safearray log (COM) 344
 - Scan for Correlations command
 - COM 355
 - Database Vusers 254
 - Script Generator, *See* VuGen
 - script view
 - Oracle NCA scripts 610
 - Web Vuser scripts 414
 - Windows Sockets scripts 275
 - Search and Replace dialog box 80
 - searching for text on screen (RTE) 673
 - sections of a Vuser script 26
 - secure WAP 748
 - SED utility 402
 - Select or Create Parameter dialog box 78
 - Select Results Directory dialog box 146
 - sequential parameter assignment 97
 - serialization (Java correlation) 194
 - Serialization options 182
 - Java (Corba, RMI Java) 181

Creating Vuser Scripts

- Service Step Properties dialog box 513
- Service steps
 - modifying in tree view (Web) 513
- session step
 - integrating Vuser scripts into 149
- settings, *See* Run-Time settings
- show function 22
- Siebel
 - base 36 key values 799
 - scripting tips for 2-tier 796
- Siebel (all types) 231
- SMS - Short Message Service 755
- SMTP protocol 684
- snapshot
 - of buffer, Winsock 278
 - Web page 528
 - XML 548
- SOAP Vuser Scripts 417
- Solaris
 - ASCII translations 269
- Speed Simulation settings 466
- split actions 45
- stand-alone mode, running Vuser scripts 139
- standard log option 127
- Start Recording dialog box
 - all 31
- Start Transaction dialog box 61
- Step button 143
- stopping a Vuser 142
- streaming data protocols
 - mms functions 720
 - RealPlayer functions 719
 - recording 717
- Submit Data step 413
 - dialog box (Web) 506
 - modifying-Web 505
- Submit Form Step
 - dialog box 503
 - modifying 502
- suffix values in Siebel 796
- Support Online xx
- synchronization functions (Baan) 626
- synchronizing Vuser scripts
 - block-mode (IBM) terminals 664
 - character-mode (VT) terminals 668
 - overview 70

- synchronizing Vuser scripts (*cont'd*)
 - overview (RTE) 663
 - waiting for terminal to be silent 671
 - waiting for text to appear (RTE) 669
 - waiting for the cursor to appear 668
- syntax, show for function 22
- system variables
 - RTE 669
 - Tuxedo 707

T

- table icon 80
- te (RTE) functions 638
- TE system variables 669
- template
 - creating new 29
 - Java Vuser 314
 - programming in "C" 764, 844
- Terminal Emulation 643
- terminal server session 227
- Terminal Setup dialog box 646
- test results 555
- TestDirector
 - connecting to TestDirector 151
 - disconnecting from TestDirector 153
 - managing Vuser scripts 151
 - opening a Vuser script 154
 - saving scripts to TestDirector 156
- text
 - comparing in snapshots (Web) 572
 - reading text from screen (RTE) 674
 - searching for text on screen (RTE) 673
- Text Check Properties dialog box 481
- text checks
 - defined 479
 - defining additional properties 488
- text view (WinSock) 278
- text view, *See* script view 414
- think time
 - defined 129
 - dialog box (Web treeview) 513
 - function (C) 21
 - function (Java) 319
 - modifying in Web scripts 512
 - recording (Web, Wireless) 436

- think time (*cont'd*)
 - run-time settings 129
 - threshold, Database 235
 - threshold, WinSock 270
- thread, main (Java programming) 329
- thread-safe code 328
- timeout, setting default (Baan) 632
- timestamp (Database) 236
- tips
 - Database related 791
 - Siebel specific 796
- token, parameterizing 519
- traffic forwarding 52
- transactions
 - automatic, for LRD functions 235
 - automatic, for Web Vuser scripts 137
 - for Web Vusers 137
 - inserting 61
- Translation table settings 269
- translation, ASCII on UNIX 269
- trapping 423
- treeview
 - Citrix scripts 218
 - Oracle NCA scripts 610
 - Web Vuser scripts 412
 - Windows Sockets scripts 275
- troubleshooting
 - 2-tier database 791
 - Oracle applications 787
 - Siebel protocol 796
 - VuGen 783
 - Web Vuser scripts 561
- TUXDIR environment setting 707
- Tuxedo Vuser scripts 697
 - data buffers 706
 - environment settings 707
 - log file 705
 - running 705
 - Tuxedo 6, 7 697
 - understanding 704
 - using lrt (Tuxedo) functions 699
 - viewing data files 706
- types of parameters, understanding 83
- typing style (RTE Vusers) 655

U

- undo buffer, emptying (WinSock) 284
- Undo Parameter command 82
- unique column name 796
- unique number, parameter values 88
- unique value parameter assignment 98
- UNIX
 - command line 148
- update methods, in parameterization 93, 97
- URL Step Properties dialog box
 - Web 498
- URL steps
 - modifying 497
- URL-based mode 445
- Use Existing Parameter command 81
- user-agent browser emulation 462

V

- VB Vusers 310
- VBA references 138
- VBA run-time setting 137
- VBScript Vusers 311
- verification checks
 - RTE 673
 - Web 467
- Virtual User Generator, *See* VuGen
- Visual Basic
 - scripting option 44
 - Vuser scripts 763
- Visual C, using Visual Studio 763
- Visual Log options
 - setting (Web) 141
 - viewing log (Web) 554
- Visual Studio 763
- VM (virtual machine) 174, 176
- VoiceXML
 - overview 729
 - See Also* Wireless entries
- VuGen
 - General recording options 49
 - overview 11
 - recording Vuser scripts 12, 25
 - running Vuser scripts 14
 - Script recording options 43
 - toolbar 36

Creating Vuser Scripts

Vuser functions

- Baan 624
 - ctx (Citrix) 216
 - external, user defined 839
 - imap 679
 - Java 317
 - lr (C functions) 15
 - lrd (Database) 239
 - lreal 719
 - lrs (WinSock) 271
 - lrt (Tuxedo) 699
 - mapi 681
 - mms (MS Media Player) 720
 - Oracle NCA 605
 - pop3 683
 - smtp 684
 - te (RTE) 638
- See Also* Online Function Reference

Vuser Generator, *See* VuGen

Vuser ID, parameter values 91

Vuser information, obtaining 65

Vuser information, obtaining (Java) 322

Vuser scripts 26

- adding functions 59
- comments, inserting 64
- Corba-Java 371
- custom 305
- debugging features 143
- development steps 6
- EJB testing 575
- enhancing 59
- Jacada 689
- Java Vuser (programming) 313
- parameterizing 75
- programming 305
- programming on UNIX 843
- regenerating 40
- rendezvous points 63
- running 139
- running from command prompt 147
- run-time settings 115
- sections 26
- selecting generation language 43
- session step integration 149
- TestDirector integration 151
- tools for developing 7

Vuser scripts (*cont'd*)

- transactions 61
- UNIX, compiling on 852
- UNIX, creating script 843
- UNIX, running on 148
- vuser_end section of Vuser script 26
- vuser_init section of Vuser script 26
- Vusers
 - introducing 3
 - types of 5

W

waiting for terminal to stabilize 671

WAP Vuser scripts

- debug information 471
- gateway options 753
- introducing 739
- run-time settings 751
- See Also* Wireless scripts
- specifying what to record 748
- Toolkit tab 750
- understanding 739

Wdiff 111

Web correlation 515

Web functions, using 405

Web performance graphs

- generating for Web Vusers 467

Web to Java converter 402

Web trapping 423

Web Trapping tab 424

Web Vuser scripts

- adding steps 494
- advanced tips 561
- checks 477
- content filtering 439
- correlating 517
- custom headers 437
- custom request steps 508
- debugging features, enabling 145
- debugging tools 144
- deleting steps 496
- functions 403
- Internet recording options 431
- introducing 397
- modifying 493

- Web Vuser scripts (*cont'd*)
 - proxy settings 432
 - recording options 443
 - regular expressions in text checks 489
 - Results Summary report 553
 - run-time viewer 144
 - sections 401
 - setting Visual Log options 144
 - specifying a browser for recording 444
 - understanding 399
 - verifying text and images 477
 - Visual Log options 141
- Web/WinSock Vuser scripts 417
 - dual vs. multi 418
 - getting started 419
 - proxy settings 421
 - recording 425
 - Web trapping options 423
- Windows Sockets Vuser scripts
 - data buffers 290
 - data files 291
 - excluding sockets 269
 - getting started 266
 - recording 265
 - script and tree view 266
 - using lrs functions 271
 - viewing data files 289
 - working with data 277
- WinInet engine (Internet protocols) 468
- WinSock recording options 268
- Wireless Vuser scripts
 - custom headers 437
 - Internet recording options 431
 - introducing 725
 - proxy settings 432
 - recording 733
 - recording options 747
 - understanding 725
 - WAP toolkit 750
- WSP
 - recording options 748
 - running mode 752
 - session-recording over a phone 740
- WSxxx Tuxedo variables 707

X

XML

- attributes 777
- custom requests 549
- testing 547

X-SYSTEM message (RTE) 664**Z**

- zip file options 37

C-Interpreter Copyright Agreement

The Virtual User Generator generates standard C code which can be compiled with any ANSI C compiler. However, for the convenience of our customers we have provided a C Interpreter for running the generated code, without charge. The cci executable which is the front end of the interpreter is based on the freely available "lcc Retargetable C Compiler" by Christopher Fraser and David Hanson, and is covered by the lcc Copyright included below. Any bugs in cci should be reported to Mercury Interactive. The author's copyright notice is below.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR MERCURY INTERACTIVE MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

The authors of this software are Christopher W. Fraser and David R. Hanson.

Copyright (c) 1991,1992,1993,1994,1995 by AT&T, Christopher W. Fraser, and David R. Hanson. All Rights Reserved.

Permission to use, copy, modify, and distribute this software for any purpose, subject to the provisions described below, without fee is hereby granted, provided that this entire notice is included in all copies of any software that is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

lcc is not public-domain software, shareware, and it is not protected by a 'copyleft' agreement, like the code from the Free Software Foundation. lcc is available free for your personal research and instructional use under the 'fair use' provisions of the copyright law. You may, however, redistribute the lcc in whole or in part provided you acknowledge its source and include this COPYRIGHT file.

You may not sell lcc or any product derived from it in which it is a significant part of the value of the product. Using the lcc front end to build a C syntax checker is an example of this kind of product.

You may use parts of lcc in products as long as you charge for only those components that are entirely your own and you acknowledge the use of lcc clearly in all product documentation and distribution media. You must state clearly that your product uses or is based on parts of lcc and that lcc is available free of charge. You must also request that bug reports on your product be reported to you. Using the lcc front end to build a C compiler for the Motorola 88000 chip and charging for and distributing only the 88000 code generator is an example of this kind of product. Using parts of lcc in other products is more problematic. For example, using parts of lcc in a C++ compiler could save substantial time and effort and therefore contribute significantly to the profitability of the product. This kind of use, or any use where others stand to make a profit from what is primarily our work, is subject to negotiation.

Chris Fraser / cwf@research.att.com David Hanson / drh@cs.princeton.edu



MERCURY INTERACTIVE

Mercury Interactive Corporation

1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Main Telephone: (408) 822-5200

Sales & Information: (800) TEST-911, (866) TOPAZ-4U

Customer Support: (877) TEST-HLP

Fax: (408) 822-5300

Home Page: www.mercuryinteractive.com

Customer Support: support.mercuryinteractive.com



PTVGUG1. 5/ 01