# OPTIMIZE

## MERCURY BUSINESS AVAILABILITY CENTER™

### Using Mercury Virtual User Generator

MERCURY™

BUSINESS TECHNOLOGY OPTIMIZATION

# Mercury Business Availability Center

## Using Mercury Virtual User Generator

Version 6.1

Document Release Date: February 28, 2006

MERCURY™

Mercury Business Availability Center, Version 6.1
Using Mercury Virtual User Generator

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: United States: 5,511,185; 5,657,438; 5,701,139; 5,870,559; 5,958,008; 5,974,572; 6,137,782; 6,138,157; 6,144,962; 6,205,122; 6,237,006; 6,341,310; 6,360,332, 6,449,739; 6,470,383; 6,477,483; 6,549,944; 6,560,564; 6,564,342; 6,587,969; 6,631,408; 6,631,411; 6,633,912; 6,694,288; 6,738,813; 6,738,933; 6,754,701; 6,792,460 and 6,810,494. Australia: 763468 and 762554. Other patents pending. All rights reserved.

Mercury, Mercury Interactive, the Mercury logo, the Mercury Interactive logo, LoadRunner, WinRunner, SiteScope and TestDirector are trademarks of Mercury Interactive Corporation and may be registered in certain jurisdictions. The absence of a trademark from this list does not constitute a waiver of Mercury's intellectual property rights concerning that trademark.

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

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

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

© 2006 Mercury Interactive Corporation, All rights reserved

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

AMLIB_VUGEN6.1/01

# Table of Contents

**Welcome to the Mercury Virtual User Generator ..... xxi**
How This Guide is Organized.....xxi
Who Should Read This Guide ..... xxiii
Getting More Information ..... xxiii

**PART I: INTRODUCING VUSER SCRIPTS**

**Chapter 1: Developing Vuser Scripts .....3**
Introducing Vusers .....4
Looking at Vuser Types .....7
The Steps of Creating Vuser Scripts.....9
Using this Guide.....10

**PART II: WORKING WITH VUGEN**

**Chapter 2: Introducing VuGen.....13**
About VuGen.....13
Starting VuGen.....14
Understanding the VuGen Environment Options .....15
Setting the Environment Options.....16
Viewing and Modifying Vuser Scripts.....17
Running Vuser Scripts with VuGen .....28
Understanding VuGen Code.....29
Using C Vuser Functions.....32
Getting Help on Functions.....36

**Chapter 3: Using the Workflow Wizard.....41**
About the Workflow Wizard .....41
Viewing the Task Pane .....42
Recording Steps .....43
Verifying the Script .....44
Enhancing the Script.....46
Prepare for Load .....52
Finishing Your Script.....52

**Chapter 4: Recording with VuGen ..... 53**
About Recording with VuGen ..... 54
Vuser Script Sections ..... 54
Creating New Virtual User Scripts ..... 56
Adding and Removing Protocols ..... 59
Choosing a Virtual User Category ..... 60
Creating a New Script ..... 61
Opening an Existing Script ..... 62
Recording Your Application ..... 63
Ending and Saving a Recording Session ..... 67
Viewing the Recording Logs ..... 69
Using Zip Files ..... 70
Importing Actions ..... 71
Providing Authentication Information ..... 72
Regenerating a Vuser Script ..... 74

**Chapter 5: Setting Script Generation Preferences ..... 77**
About Setting Script Generation Preferences ..... 77
Selecting a Script Language ..... 78
Applying the Basic Options ..... 78
Understanding the Correlation Options ..... 80
Setting Script Recording Options ..... 81

**Chapter 6: Configuring the Port Mappings ..... 83**
About Configuring the Port Mappings ..... 84
Defining Port Mappings ..... 84
Adding a New Server Entry ..... 86
Setting the Auto-Detection Options ..... 88
Setting the Port Mapping Recording Options ..... 90

**Chapter 7: Enhancing Vuser Scripts ..... 93**
About Enhancing Vuser Scripts ..... 94
Inserting Transactions into a Vuser Script ..... 96
Inserting Rendezvous Points (LoadRunner and Tuning only) ..... 99
Inserting Comments into a Vuser Script ..... 100
Obtaining Vuser Information ..... 101
Sending Messages to Output ..... 101
Handling Errors in Vuser Scripts During Execution ..... 105
Synchronizing Vuser Scripts ..... 107
Emulating User Think Time ..... 107
Handling Command Line Arguments ..... 108
Encrypting Text ..... 109
Encoding Passwords Manually ..... 110
Adding Files to the Script Folder ..... 111

**Chapter 8: Working with VuGen Parameters ..... 113**
About VuGen Parameters ..... 114
Understanding Parameter Limitations ..... 115
Creating Parameters ..... 116
Understanding Parameter Types ..... 119
Defining Parameter Properties ..... 122
Using Existing Parameters ..... 124
Using the Parameter List ..... 127
Setting Parameterization Options ..... 129

**Chapter 9: File and Table Type Parameters ..... 133**
Selecting or Creating Data Files or Data Tables ..... 134
Setting Properties for File Type Parameters ..... 140
Setting Properties for Table Type Parameters ..... 142
Choosing Data Assignment Methods for File/Table Type Parameters 144

**Chapter 10: Setting Parameter Properties ..... 149**
About Setting Parameter Properties ..... 149
Setting Properties for Internal Data Parameter Types ..... 150
Setting Properties for User-Defined Functions ..... 160
Customizing Parameter Formats ..... 161
Selecting an Update Method ..... 162

**Chapter 11: Correlating Statements ..... 165**
About Correlating Statements ..... 165
Using Correlation Functions for C Vusers ..... 167
Using Correlation Functions for Java Vusers ..... 169
Comparing Vuser Scripts using WDiff ..... 170
Modifying Saved Parameters ..... 172

**Chapter 12: Configuring Run-Time Settings ..... 173**
About Run-Time Settings ..... 174
Configuring Run Logic Run-Time Settings (multi-action) ..... 175
Pacing Run-Time Settings ..... 180
Configuring Pacing Run-Time Settings (multi-action) ..... 182
Setting Pacing and Run Logic Options (single action) ..... 183
Configuring the Log Run-Time Settings ..... 184
Configuring the Think Time Settings ..... 189
Configuring Additional Attributes Run-Time Settings ..... 191
Configuring Miscellaneous Run-Time Settings ..... 192
Setting the VB Run-Time Settings ..... 198

**Chapter 13: Configuring Network Run-Time Settings ..... 201**
About Network Run-Time Settings ..... 201
Setting the Network Speed ..... 202

**Chapter 14: Running Vuser Scripts in Standalone Mode .....203**
About Running Vuser Scripts in Standalone Mode .....204
Running a Vuser Script in VuGen .....204
Replaying a Vuser Script .....207
Using VuGen's Debugging Features .....210
Using VuGen's Debugging Features for Web Vuser Scripts .....215
Working with VuGen Windows .....217
Running a Vuser Script from a Command Prompt .....217
Running a Vuser Script from a UNIX Command Line .....218
Integrating Scripts into Tests .....221

**Chapter 15: Managing Scripts Using Quality Center .....225**
About Managing Scripts Using Quality Center .....225
Connecting to and Disconnecting from Quality Center .....226
Opening Scripts from a Quality Center Project .....229
Saving Scripts to a Quality Center Project .....231
Managing Script Versions in VuGen .....232

**Chapter 16: Managing Scripts with Performance Center .....241**
About Managing Scripts with Performance Center .....241
Connecting VuGen to Performance Center .....242
Uploading Vuser Scripts .....244
Downloading Vuser Scripts .....249

**PART III: WORKING WITH JAVA LANGUAGE PROTOCOLS**

**Chapter 17: Recording Java Language Vuser Scripts .....255**
About Recording Java Language Vuser Scripts .....256
Getting Started with Recording .....256
Understanding Java Language Vuser Scripts .....258
Running a Script as Part of a Package .....258
Viewing the Java Methods .....259
Manually Inserting Java Methods .....261
Configuring Script Generation Settings .....263

**Chapter 18: Setting Java Recording Options .....267**
About Setting Java Recording Options .....268
Java Virtual Machine (JVM) Recording Options .....269
Setting Classpath Recording Options .....271
Recorder Options .....272
Serialization Options .....276
Correlation Options .....277
Debug Options .....279
CORBA Options .....282

**Chapter 19: Correlating Java Scripts.....285**
About Correlating Java Scripts .....286
Standard Correlation .....287
Advanced Correlation .....287
String Correlation .....289
Using the Serialization Mechanism .....290

**Chapter 20: Configuring Java Run-Time Settings .....297**
About Configuring Java Run-Time Settings .....297
Specifying the JVM Run-Time Settings .....298
Setting the Run-Time Classpath Options .....299

**PART IV: APPLICATION DEPLOYMENT SOLUTION PROTOCOLS**

**Chapter 21: Creating Citrix Vuser Scripts.....303**
About Creating Citrix Vuser Scripts .....304
Getting Started with Citrix Vuser Scripts .....305
Setting Up the Client and Server .....306
Recording Tips .....309
Understanding Citrix Recording Options .....311
Setting the Citrix Recording Options .....316
Setting the Citrix Display Settings .....317
Setting the Citrix Run-Time Settings .....318
Viewing and Modifying Citrix Vuser Scripts .....321
Synchronizing Replay .....322
Understanding ICA Files .....328
Using Citrix Functions .....329
Tips for Replaying and Troubleshooting Citrix Vuser Scripts .....333

**Chapter 22: Using the LoadRunner Citrix Agent.....337**
About the LoadRunner Citrix Agent .....337
Benefitting From the Citrix Agent .....338
Installation .....343
Effects and Memory Requirements of the Citrix Agent .....344
Sample Script .....344

## PART V: CLIENT SERVER PROTOCOLS

**Chapter 23: Developing Database Vuser Scripts ..... 347**
About Developing Database Vuser Scripts ..... 348
Introducing Database Vusers ..... 349
Understanding Database Vuser Technology ..... 350
Getting Started with Database Vuser Scripts ..... 351
Setting Database Recording Options ..... 352
Database Advanced Recording Options ..... 354
Using LRD Functions ..... 356
Understanding Database Vuser Scripts ..... 362
Evaluating Error Codes ..... 367
Handling Errors ..... 368

**Chapter 24: Correlating Database Vuser Scripts ..... 371**
About Correlating Database Vuser Scripts ..... 371
Scanning a Script for Correlations ..... 372
Correlating a Known Value ..... 374
Database Correlation Functions ..... 376

**Chapter 25: Developing DNS Vuser Scripts ..... 377**
About Developing DNS Vuser Scripts ..... 377
Working with DNS Functions ..... 378

**Chapter 26: Developing WinSock Vuser Scripts ..... 379**
About Recording Windows Sockets Vuser Scripts ..... 379
Getting Started with Windows Sockets Vuser Scripts ..... 380
Setting the WinSock Recording Options ..... 382
Using LRS Functions ..... 385

**Chapter 27: Working with Windows Socket Data ..... 389**
About Working with Windows Socket Data ..... 390
Viewing Data in the Snapshot Window ..... 390
Navigating Through the Data ..... 392
Modifying Buffer Data ..... 395
Modifying Buffer Names ..... 401
Viewing Windows Socket Data in Script View ..... 402
Understanding the Data File Format ..... 404
Viewing Buffer Data in Hexadecimal format ..... 406
Setting the Display Format ..... 408
Debugging Tips ..... 411
Manually Correlating WinSock Scripts ..... 412

## PART VI: CUSTOM VUSER SCRIPTS

**Chapter 28: Creating Custom Vuser Scripts ..... 417**
About Creating Custom Vuser Scripts ..... 418
C Vusers ..... 419
Using the Workflow Wizard for C Vuser Scripts ..... 420
Java Vusers ..... 423
VB Vusers ..... 424
VBScript Vusers ..... 425
JavaScript Vusers ..... 426

**Chapter 29: Programming Java Scripts ..... 427**
About Programming Java Scripts ..... 428
Creating a Java Vuser ..... 429
Editing a Java Vuser Script ..... 429
Java Vuser API Functions ..... 432
Working with Java Vuser Functions ..... 435
Setting your Java Environment ..... 441
Running Java Vuser Scripts ..... 442
Compiling and Running a Script as Part of a Package ..... 443
Programming Tips ..... 444

## PART VII: DISTRIBUTED COMPONENT PROTOCOLS

**Chapter 30: Recording COM Vuser Scripts ..... 449**
About Recording COM Vuser Scripts ..... 450
COM Overview ..... 450
Getting Started with COM Vusers ..... 452
Selecting COM Objects to Record ..... 453
Setting COM Recording Options ..... 456

**Chapter 31: Understanding COM Vuser Scripts ..... 465**
About COM Vuser Scripts ..... 465
Understanding VuGen COM Script Structure ..... 466
Examining Sample VuGen COM Scripts ..... 468
Scanning a Script for Correlations ..... 474
Correlating a Known Value ..... 476

**Chapter 32: Understanding COM Vuser Functions ..... 479**
About COM Vuser Functions ..... 480
Creating Instances ..... 480
IDispatch Interface Invoke Method ..... 481
Type Assignment Functions ..... 481
Variant Types ..... 482
Assignment from Reference to Variant ..... 483
Parameterization Functions ..... 484
Extraction from Variants ..... 486
Assignment of Arrays to Variants ..... 486
Array Types and Functions ..... 487
Byte Array Functions ..... 488
ADO RecordSet Functions ..... 489
Debug Functions ..... 489
VB Collection Support ..... 489

**Chapter 33: Developing Corba-Java Vuser Scripts ..... 491**
About Developing Corba-Java Vuser Scripts ..... 491
Recording a Corba-Java Vuser ..... 492
Working with Corba-Java Vuser Scripts ..... 496
Recording on Windows XP and Windows 2000 Servers ..... 498
Application Specific Tips ..... 500

**Chapter 34: Developing RMI-Java Vuser Scripts ..... 501**
About Developing RMI-Java Vuser Scripts ..... 501
Recording RMI over IIOP ..... 502
Recording an RMI Vuser ..... 503
Working with RMI Vuser Scripts ..... 506

**PART VIII: E-BUSINESS PROTOCOLS**

**Chapter 35: Developing FTP Vuser Scripts ..... 511**
About Developing FTP Vuser Scripts ..... 511
Working with FTP Functions ..... 512

**Chapter 36: Developing LDAP Vuser Scripts ..... 515**
About Developing LDAP Vuser Scripts ..... 515
Working with LDAP Functions ..... 516
Defining Distinguished Name Entries ..... 519

**Chapter 37: Recording Microsoft .NET Vuser Scripts.....521**
About Recording Microsoft .NET Vuser Scripts.....522
Getting Started with Microsoft .NET Vusers.....523
Setting Microsoft .NET Recording Options.....524
Viewing Scripts in VuGen and Visual Studio.....525
Adding .NET References in the Run-Time Settings.....528
Viewing Data Sets and Grids.....530
Troubleshooting Your Script.....531
Correlating Microsoft .NET Scripts.....535

**Chapter 38: Creating Web Vuser Scripts.....537**
About Developing Web Vuser Scripts.....537
Introducing Web Vusers.....538
Understanding Web Vuser Technology.....539
Getting Started with Web Vuser Scripts.....540
Recording a Web Session.....542
Converting Web Vuser Scripts into Java.....544

**Chapter 39: Using Web Vuser Functions.....545**
About Web Vuser Functions.....545
Adding and Editing Functions.....546
Web Function List.....548
Improving Performance Using Caching.....555

**Chapter 40: Setting Recording Options for Internet Protocols.....559**
About Setting Recording Options for Internet Protocols.....559
Working with Proxy Settings.....560
Setting Advanced Recording Options.....564
Setting a Recording Scheme.....566

**Chapter 41: Setting Recording Options for Web Vusers.....573**
About Setting Recording Options.....573
Specifying which Browser to Use for Recording.....574
Selecting a Recording Level.....575

**Chapter 42: Configuring Internet Run-Time Settings.....591**
About Internet Run-Time Settings.....591
Setting Proxy Options.....593
Setting Browser Emulation Properties.....598
Setting Internet Preferences.....602
Filtering Web Sites.....608
Obtaining Debug Information.....610
Performing HTML Compression.....611

**Chapter 43: Checking Web Page Content ..... 613**
About Checking Web Page Content ..... 613
Setting the ContentCheck Run-Time Settings ..... 614

**Chapter 44: Verifying Web Pages Under Load ..... 619**
About Verification Under Load ..... 619
Adding a Text Check ..... 622
Understanding Text Check Functions ..... 624
Adding an Image Check ..... 630
Defining Additional Properties ..... 633

**Chapter 45: Modifying Web and Wireless Vuser Scripts ..... 635**
About Modifying Web and Wireless Vuser Scripts ..... 636
Adding a Step to a Vuser Script ..... 637
Deleting Steps from a Vuser Script ..... 638
Modifying Action Steps ..... 639
Modifying Control Steps ..... 656
Modifying Service Steps ..... 659
Modifying Web Checks (Web only) ..... 660

**Chapter 46: Setting Correlation Rules for Web Vuser Scripts ..... 661**
About Correlating Statements ..... 662
Understanding the Correlation Methods ..... 663
Using VuGen's Correlation Rules ..... 664
Setting Correlation Rules ..... 669
Testing Rules ..... 672
Setting the Correlation Recording Options ..... 673

**Chapter 47: Correlating Vuser Scripts After Recording ..... 675**
About Correlating with Snapshots ..... 676
Viewing the Correlation Results Tab ..... 677
Setting Up VuGen for Correlations ..... 680
Performing a Scan for Correlations ..... 683
Performing Manual Correlation ..... 687
Defining a Dynamic String's Boundaries ..... 692

**Chapter 48: Testing XML Pages ..... 695**
About Testing XML Pages ..... 695
Viewing XML as URL Steps ..... 696
Inserting XML as a Custom Request ..... 698
Viewing XML Custom Request Steps ..... 700

**Chapter 49: Using Reports to Debug Vuser Scripts....703**
About Using Reports to Debug Vuser Scripts....704
Understanding the Results Summary Report....705
Filtering Report Information....707
Searching Your Results....708
Managing Execution Results....708

**Chapter 50: Power User Tips for Web Vusers....711**
Security Issues....711
Handling Cookies....715
The Run-Time Viewer (Online Browser)....718
Browsers....719
Configuration and Compatibility Issues....722

**Chapter 51: Planning Web Service Tests....723**
About Planning Web Service Tests....724
Implementing a Web Service....725
Challenges in Web Services Testing....729
Choosing a Web Services Script Type....730
Performing a Load Test....732
Client Emulation....733

**Chapter 52: Developing Web Services Vusers....735**
About Web Services....736
Getting Started with Web Services in VuGen....736
Using the Workflow Wizard for Web Services Scripts....738
Creating a New Web Services Script....739
Recording a Web Services Script....740
Scanning WSDL Documents....743
Managing WSDL Documents....754
Setting WSDL Validation and Comparison Options....760
Editing an XML Tree....764
IDE Integration....765

**Chapter 53: Running Web Services Vusers....767**
About Running Web Services Vusers....767
Setting Web Service Run-Time Settings....768
Comparing WSDL Files....769
Comparing XML Files....773
Setting Scanned WSDL Properties....775
Viewing Web Services Script Snapshots....775
Using Web Services Functions....783
Viewing Web Services Reports....785

**Chapter 54: Recording Web/WinSock Vuser Scripts ..... 789**
About Recording Web/WinSock Vuser Scripts ..... 790
Getting Started with Web/WinSock Vuser Scripts ..... 791
Setting Browser and Proxy Recording Options ..... 792
Setting Web Trapping Recording Options ..... 796
Recording a Web/WinSock Session ..... 798
Recording Palm Applications ..... 800

**PART IX: ENTERPRISE JAVA BEAN PROTOCOLS**

**Chapter 55: Performing EJB Testing ..... 805**
About EJB Testing ..... 805
Working with the EJB Detector ..... 806
Creating an EJB Testing Vuser ..... 811
Setting EJB Recording Options ..... 815
Understanding EJB Vuser Scripts ..... 816
Running EJB Vuser Scripts ..... 822

**PART X: ERP/CRM PROTOCOLS**

**Chapter 56: Developing Web GUI-Level Scripts ..... 829**
About Developing Web GUI-Level Scripts ..... 830
Getting Started with Web GUI-Level Vusers ..... 831
Using GUI-Level Vuser Functions ..... 832
Understanding GUI-level Vuser Scripts ..... 835
Tips for Working with the GUI-Level Vusers ..... 838

**Chapter 57: Setting Web GUI Recording Options ..... 839**
About Setting Web GUI Recording Options ..... 839
Configuring Object Identification ..... 840
Configuring Web Event Recording ..... 846
Selecting a Recording Level for GUI-Level Vusers ..... 859

**Chapter 58: Creating Oracle NCA Vuser Scripts .....867**
About Creating Oracle NCA Vuser Scripts .....868
Getting Started with Oracle NCA Vusers .....869
Recording Guidelines .....870
Enabling the Recording of Objects by Name .....872
Oracle Applications via the Personal Home Page .....876
Using Oracle NCA Vuser Functions .....877
Understanding Oracle NCA Vusers .....881
Configuring Oracle NCA Run-Time Settings .....883
Testing Oracle NCA Applications .....886
Correlating Oracle NCA Statements for Load Balancing .....889
Additional Recommended Correlations .....890
Recording in Pragma Mode .....893

**Chapter 59: Developing SAPGUI Vuser Scripts .....895**
About Developing SAPGUI Vuser Scripts .....896
Checking your Environment for SAPGUI Vusers .....897
Creating a SAPGUI Vuser Script .....908
Recording a SAPGUI Vuser Script .....909
Setting the SAPGUI Recording Options .....912
Inserting Steps Interactively into a SAPGUI Script .....915
Understanding a SAPGUI Vuser Script .....917
Enhancing a SAPGUI Vuser Script .....921
Replaying SAPGUI Optional Windows .....925
Setting SAPGUI Run-Time Settings .....925
SAPGUI Functions .....929
Tips for SAPGUI Vuser Scripts .....937
Troubleshooting SAPGUI Vuser Scripts .....942
Additional Resources .....944

**Chapter 60: Developing SAP-Web Vuser Scripts .....945**
About Developing SAP-Web Vuser Scripts .....946
Creating a SAP-Web Vuser Script .....946
Setting SAP-Web Recording Options .....948
Understanding a SAP-Web Vuser Script .....949
Replaying a SAP-Web Vuser Script .....951

**Chapter 61: Developing Siebel-Web Vuser Scripts .....953**
About Developing Siebel-Web Vuser Scripts .....953
Recording a Siebel-Web Session .....954
Correlating Siebel-Web Scripts .....955
Correlating SWECount, ROWID, and SWET Parameters .....962
Troubleshooting Siebel-Web Vuser Scripts .....964

## PART XI: LEGACY PROTOCOLS

**Chapter 62: Introducing RTE Vuser Scripts .....971**
About Developing RTE Vuser Scripts .....971
Introducing RTE Vusers .....972
Understanding RTE Vuser Technology .....973
Getting Started with RTE Vuser Scripts .....973
Using TE Functions .....975
Mapping Terminal Keys to PC Keyboard Keys .....977

**Chapter 63: Recording RTE Vuser Scripts .....979**
About Recording RTE Vuser Scripts .....980
Creating a New RTE Vuser Script .....980
Recording the Terminal Setup and Connection Procedure .....981
Recording Typical User Actions .....986
Recording the Log Off Procedure .....987
Setting RTE Configuration Options .....988
Setting the RTE Recording Options .....989
Typing Input into a Terminal Emulator .....992
Generating Unique Device Names .....995
Setting the Field Demarcation Characters .....997

**Chapter 64: Configuring RTE Run-Time Settings .....999**
About Terminal Emulator Run-Time Settings .....1000
Modifying Connection Attempts .....1001
Specifying an Original Device Name .....1002
Setting the Typing Delay .....1002
Configuring the X-System Synchronization .....1003

**Chapter 65: Synchronizing RTE Vuser Scripts .....1005**
About Synchronizing Vuser Scripts .....1005
Synchronizing Block-Mode (IBM) Terminals .....1007
Synchronizing Character-Mode (VT) Terminals .....1010

**Chapter 66: Reading Text from the Terminal Screen .....1017**
About Reading Text from the Terminal Screen .....1017
Reading Text from the Screen .....1018

## PART XII: MAILING SERVICES PROTOCOLS

**Chapter 67: Developing Vuser Scripts for Mailing Services ..... 1023**
About Developing Vuser Scripts for Mailing Services ..... 1023
Getting Started with Mailing Services Vuser Scripts ..... 1024
Working with IMAP Functions ..... 1026
Working with MAPI Functions ..... 1028
Working with POP3 Functions ..... 1030
Working with SMTP Functions ..... 1031

## PART XIII: MIDDLEWARE PROTOCOLS

**Chapter 68: Developing Jacada Vuser Scripts ..... 1035**
About Jacada Vuser Scripts ..... 1035
Getting Started with Jacada Vusers ..... 1036
Recording a Jacada Vuser ..... 1038
Replaying a Jacada Vuser ..... 1040
Understanding Jacada Vuser Scripts ..... 1041
Working with Jacada Vuser Scripts ..... 1042

**Chapter 69: Developing Tuxedo Vuser Scripts ..... 1045**
About Tuxedo Vuser Scripts ..... 1046
Getting Started with Tuxedo Vuser Scripts ..... 1047
Using LRT Functions ..... 1048
Understanding Tuxedo Vuser Scripts ..... 1053
Viewing Tuxedo Buffer Data ..... 1055
Defining Environment Settings for Tuxedo Vusers ..... 1057
Debugging Tuxedo Applications ..... 1058
Correlating Tuxedo Scripts ..... 1058

## PART XIV: STREAMING DATA PROTOCOLS

**Chapter 70: Developing Streaming Data Vuser Scripts ..... 1067**
About Recording Streaming Data Virtual User Scripts ..... 1068
Getting Started with Streaming Data Vuser Scripts ..... 1068
Using RealPlayer LREAL Functions ..... 1070
Using Media Player MMS Functions ..... 1071

## PART XV: WIRELESS PROTOCOLS

**Chapter 71: Introducing Wireless Vusers ..... 1075**
About Wireless Vusers ..... 1075
Understanding the WAP Protocol ..... 1076
Understanding the i-mode System ..... 1078
i-mode versus WAP ..... 1079
Understanding VoiceXML ..... 1080

**Chapter 72: Recording Wireless Vuser Scripts ..... 1083**
About Recording Wireless Vuser Scripts ..... 1083
Getting Started with Wireless Vuser Scripts ..... 1084
Using Wireless Vuser Functions ..... 1086
Troubleshooting Wireless Vuser Scripts ..... 1087

**Chapter 73: Working with WAP Vuser Scripts ..... 1089**
About WAP Vusers ..... 1089
Recording Over a Phone ..... 1090
Bearers Support ..... 1091
RADIUS Support ..... 1092
Push Support ..... 1092
VuGen Push Support ..... 1094

**Chapter 74: Setting Wireless Vuser Recording Options ..... 1097**
About Setting Wireless Recording Options ..... 1097
Specifying the Recording Mode (WAP only) ..... 1098
Specifying the Information to Record (i-mode and VoiceXML) ..... 1099
Specifying a Toolkit ..... 1101

**Chapter 75: Configuring WAP Run-Time Settings ..... 1105**
About WAP Run-Time Settings ..... 1105
Configuring Gateway Options ..... 1106
Configuring Radius Connection Data ..... 1111

## PART XVI: INFORMATION FOR ADVANCED USERS

**Chapter 76: Creating Vuser Scripts in Visual Studio ..... 1115**
About Creating Vuser Scripts in Visual Studio ..... 1115
Creating a Vuser Script with Visual C ..... 1116
Creating a Vuser Script with Visual Basic ..... 1118
Configuring Runtime Settings and Parameters ..... 1120

**Chapter 77: Programming with the XML API ....1121**
About Programming with the XML API....1121
Understanding XML Documents ....1123
Using XML Functions....1124
Specifying XML Function Parameters ....1127
Working with XML Attributes ....1129
Structuring an XML Script....1129
Enhancing a Recorded Session ....1131

**Chapter 78: VuGen Debugging Tips....1137**
General Debugging Tip ....1138
Using C Functions for Tracing ....1138
Adding Additional C Language Keywords ....1138
Examining Replay Output....1139
Debugging Database Applications ....1139
Working with Oracle Applications....1141
Solving Common Problems with Oracle 2-Tier Vusers....1142
Two-tier Database Scripting Tips....1147
Running PeopleSoft-Tuxedo Scripts....1156

**Chapter 79: Advanced Topics ....1157**
Files Generated During Recording ....1158
Files Generated During Replay....1160
Running a Vuser from the Unix Command Line ....1162
Specifying the Vuser Behavior ....1163
Command Line Parameters....1164
Recording OLE Servers....1165
Examining the .dat Files....1167
Adding a New Vuser Type ....1168

**PART XVII: APPENDIXES**

**Appendix A: Calling External Functions....1177**
Loading a DLL—Locally ....1177
Loading a DLL—Globally ....1179

**Appendix B: Working with Foreign Languages....1181**
About Working with Foreign Languages ....1181
Manually Converting String Encoding ....1182
Converting String Encoding In Parameter Files....1183
Setting the String Encoding for Web Record and Replay ....1185
Specifying a Language for the Accept-Language Header ....1188
Protocol Limitations....1189
Quality Center Integration....1190

**Appendix C: Programming Scripts on UNIX Platforms.....1191**
About Programming Vuser Scripts to Run on UNIX Platforms.....1191
Generating Templates .....1192
Programming Vuser Actions .....1193
Configuring Vuser Run-Time Settings .....1195
Defining Transactions and Rendezvous Points.....1200
Compiling Scripts.....1200

**Appendix D: Using Keyboard Shortcuts .....1203**

**Index.....1205**

# Welcome to the Mercury Virtual User Generator

This guide describes how to work with the Mercury Virtual User Generator (VuGen) to create scripts for Business Process profiles. Scripts contain transactions, which emulate real users performing business processes in your applications.

## How This Guide is Organized

The guide contains the following parts:

**Part I Introducing Vuser Scripts is applicable to all Vuser types.**

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

**Part II Working with VuGen**

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

### Part III to XV Recording Protocols

These sections provide information on how to record Vuser scripts for various protocols. The sections include:

- Working with Java Language Protocols
- Application Deployment Solution Protocols
- Client Server Protocols
- Custom Vuser Scripts
- Distributed Component Protocols
- E-Business Protocols
- Enterprise Java Bean Protocols
- ERP/CRM Protocols
- Legacy Protocols
- Mailing Services Protocols
- Middleware Protocols
- Streaming Data Protocols
- Wireless Protocols

### Part XVI Information for Advanced Users

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

### Part XVII Appendixes

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

## Who Should Read This Guide

This guide is intended for the following users of Mercury Business Availability Center:

- Mercury Business Availability Center administrators
- Mercury Business Availability Center data collector administrators
- Script developers

Readers of this guide should be knowledgeable about enterprise system administration, scripting, and enterprise applications.

## Getting More Information

For information on using and updating the Mercury Business Availability Center Documentation Library, reference information on additional documentation resources, typographical conventions used in the Documentation Library, and quick reference information on deploying, administering, and using Mercury Business Availability Center, refer to *Getting Started with Mercury Business Availability Center*.

# Part I

## Introducing Vuser Scripts

# 1

# Developing Vuser Scripts

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

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

This chapter includes:

- Introducing Vusers
- Looking at Vuser Types
- The Steps of Creating Vuser Scripts
- Using this Guide

---

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

---

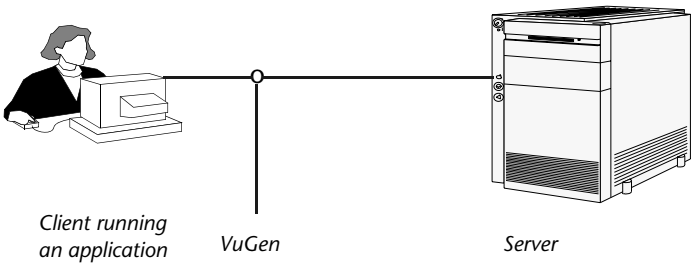
# Introducing Vusers

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

To accomplish this emulation, the human was replaced with a virtual user, or a *Vuser*. Vusers emulate the actions of human users by performing typical business processes in your application. The actions that a Vuser performs during the recording session are described in a *Vuser script*.

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

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



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



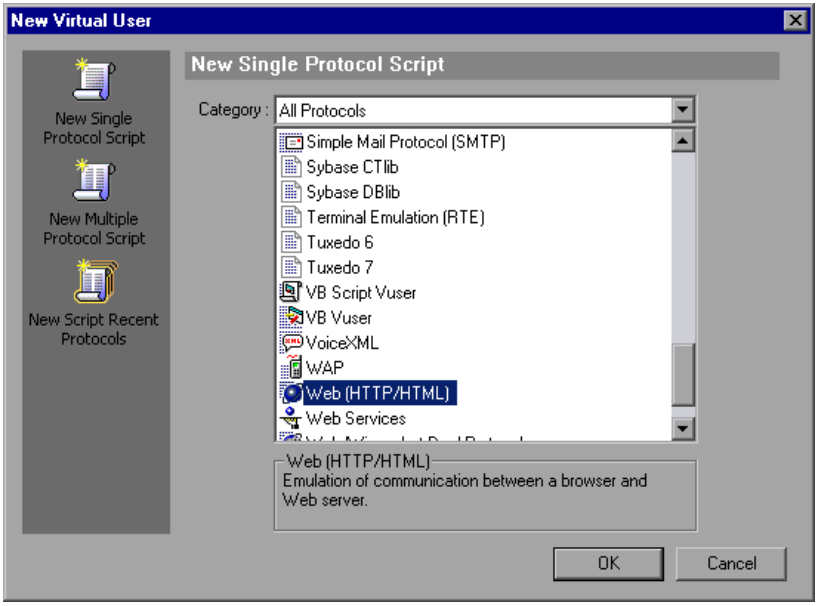
*Vuser running an a client machine*

*Server*

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

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

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




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

## Looking at Vuser Types

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

The Vuser types are divided into the following categories:

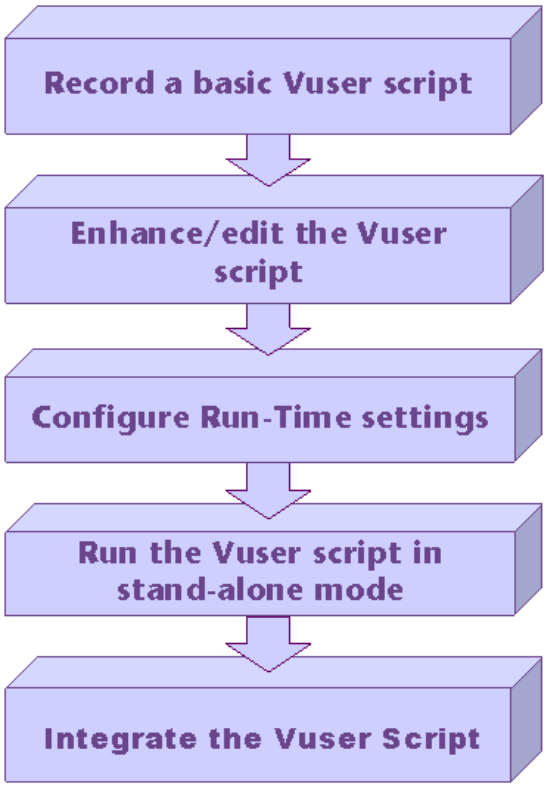
- **All Protocols:** a list of all supported protocols in alphabetical order
- **Application Deployment Solution:** For the Citrix protocol
- **Client/Server:** For DB2 CLI, DNS, MS SQL, ODBC, Oracle (2-tier), Sybase Ctlib, Sybase Dblib, and Windows Sockets protocols
- **Custom:** For C templates, Visual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

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

# The Steps of Creating Vuser Scripts

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




**The process of creating a Vuser script is as follows:**

1. Record a basic script using VuGen. If you are testing Windows-based GUI applications or complex Web environments such as applets and Flash, you may need to use Mercury's GUI-based tools such as WinRunner and QuickTest Professional.
2. Enhance the basic script by adding control-flow statements and other Mercury API functions into the script.
3. Configure the run-time settings. These settings include iteration, log, and timing information, and define the Vuser behavior during a script run.

4 Verify the script's functionality, run it in standalone mode.

5 After you verify that the script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* user guides.

## Using this Guide

This guide is divided into several parts:

- Part I, "Introducing Vuser Scripts," is applicable to all Vuser types.
- Part II, "Working with VuGen," is applicable only to those Vuser types that are recorded and/or run using VuGen. Part II is not applicable when developing GUI Vuser scripts.
- Parts III to XVI apply to specific Vuser types. Refer to the Table of Contents to locate the part describing your Vuser type.
- Part XVII 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 XVIII contains several appendixes with technology overviews. It describes the Calling External Functions, Programming in UNIX, Working with Foreign Languages, and Keyboard Shortcuts.

---

**Note:** GUI Vuser scripts utilize the WinRunner engine—you do not generate these scripts using VuGen. For more information, see "Developing GUI Vuser Scripts" in the online book, and refer to the *WinRunner User's Guide*.

---

# 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:

- About VuGen
- Starting VuGen
- Understanding the VuGen Environment Options
- Setting the Environment Options
- Viewing and Modifying Vuser Scripts
- 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 except for GUI.***

## About VuGen

The Virtual User Generator, also known as *VuGen*, is the primary tool for developing Vuser scripts.

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

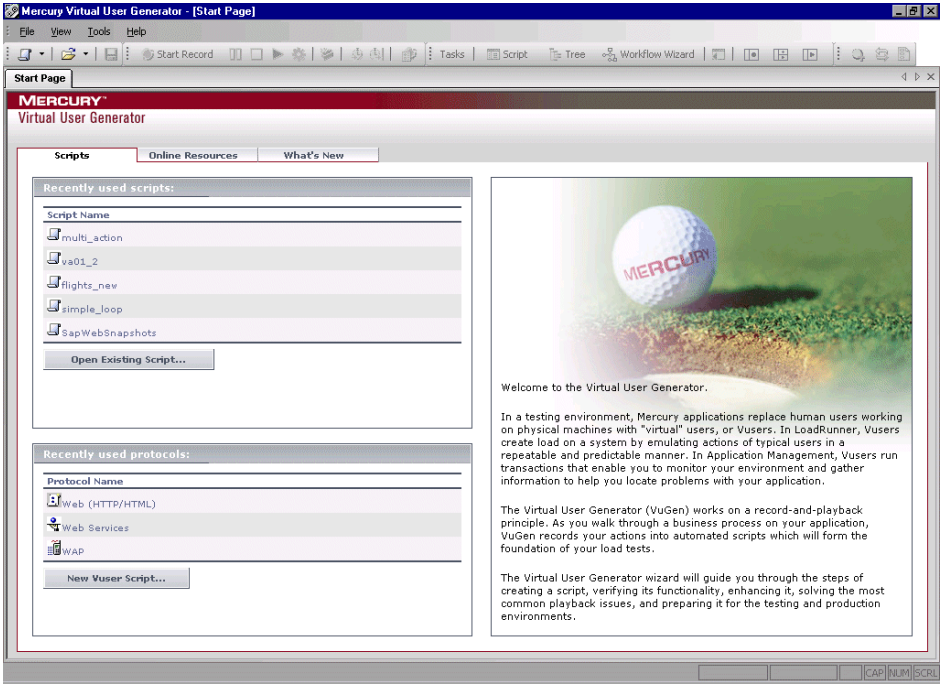
**Note:** VuGen records sessions on Windows platforms only. However, a recorded Vuser script can run on both Windows and UNIX platform.

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

## Starting VuGen

To start VuGen, choose **Start** > **Programs** > **Mercury** *App_Name* > **Applications** > **Virtual User Generator** from the Start menu.

The Virtual User Generator Start Page opens.




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

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

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

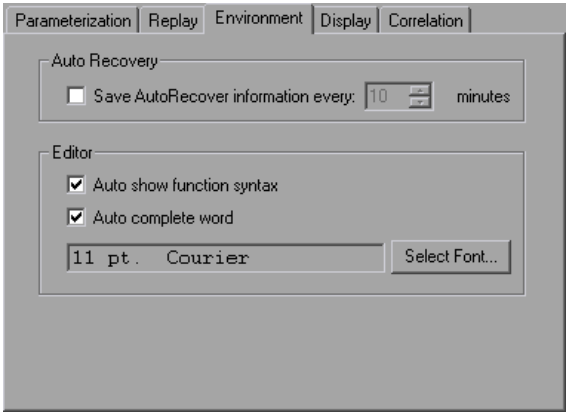
To create a new script in a protocol that is not listed, click **New Vuser Script**.

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

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

## Understanding the VuGen Environment Options

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




### Auto Recovery

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

### Editor

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

**Auto show function syntax:** When you type the opening parenthesis of a function, VuGen shows the syntax of the function with its arguments and prototypes. To enable the showing of the syntax globally, select the check box adjacent to this option. To disable this feature, clear the check box adjacent to the **Auto show function syntax** option. If you disable **Show Function Syntax** globally, you can still bring up the syntax by pressing Ctrl+Shift+Space or choosing **Edit** > **Show Function Syntax** after typing the opening parenthesis in the editor.

**Auto complete word:** When you type the first underscore of a function, VuGen opens a list of functions allowing you to choose the exact function without having to manually type in the entire function. To enable word completion globally, select the check box adjacent to this option. To disable this feature, clear the check box adjacent to the **Auto complete word** option. If you disable this option globally, you can still bring up the function list box by pressing Ctrl+Space or choosing **Edit** > **Complete Word** while typing in the editor.

**Select Font:** To set the editor font, click **Select Font**. The Font dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.

### Default Environment Settings

By default, **Show Function Syntax** and **Auto complete word** are enabled globally. Auto Recovery is set to 10 seconds.

## Setting the Environment Options

**To set the environment-related options:**

1. Select **Tools** > **General Options** and click the Environment tab.
2. To save the current script information for auto recovery, select the **Save AutoRecover Information** option and specify the time in minutes between the saves.

3 To set the editor font, click **Select Font**. The Font dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.

4 To 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.

# Viewing and Modifying Vuser Scripts

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

The Script view is available for all Vuser types, but the Tree and Thumbnail views are not. For example, Thumbnail view is only available for Web, Citrix, and SAPGUI Vusers.

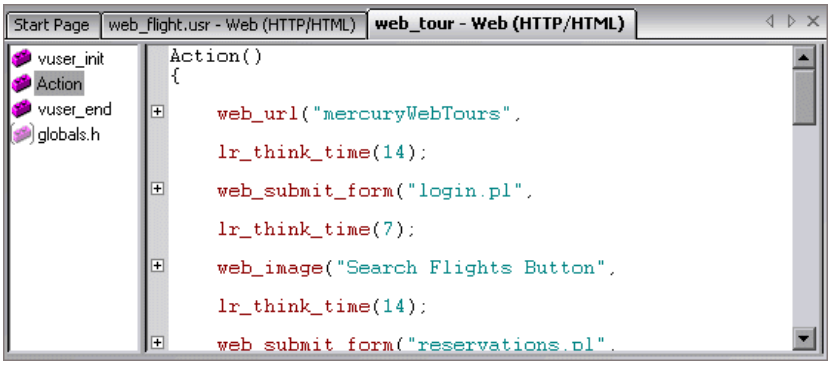
## Viewing the Code in Script View

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

**To display the script view:**



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






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

When working in Script view, you can add steps to the script using the **Insert** > **New Step** command. Alternatively, you can manually enter functions using the Complete Word and Show Function Syntax features. For more information, see “Getting Help on Functions” on page 36.

---

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

---

## Viewing a Script in Tree View

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

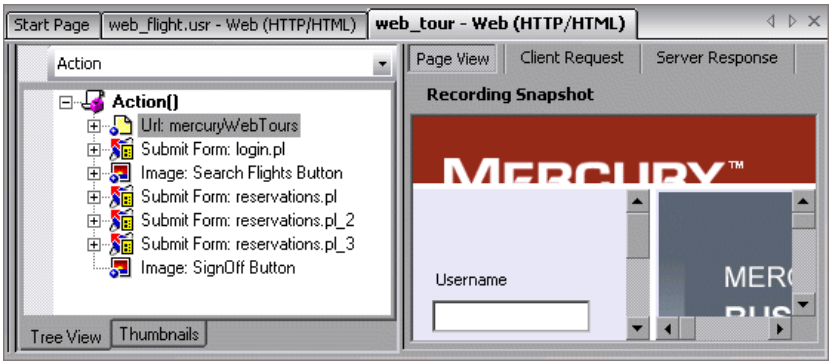
**To display the Tree view:**



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

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

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



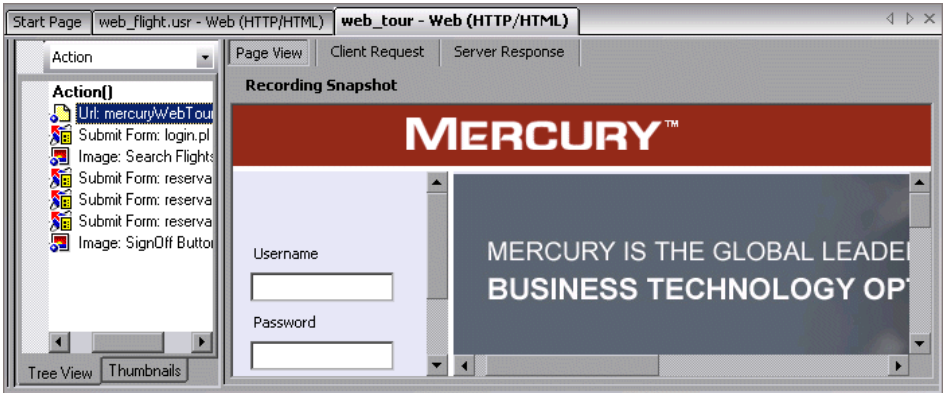

Within the Tree view, you can manipulate steps by dragging them to the desired location. You can also add additional steps between existing steps in the tree hierarchy.

**To insert a step in Tree view:**

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

## Understanding Snapshots

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




VuGen captures a base snapshot during recording and another one during replay. You compare the Record and Replay snapshots to determine the dynamic values that need to be correlated in order to run the script. For more information, see Chapter 47, "Correlating Vuser Scripts After Recording."

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



Show a full window of the recorded snapshot



Show a split window of the recorded and replayed snapshot



Show a full window of the replayed snapshot

**To view or hide snapshots:**

1 Make sure you are in Tree view. If not, then switch to Tree view (**View** > **Tree View**).

2 Choose **View** > **Snapshot** > **View Snapshot**. VuGen shows the snapshot of the client window. If the snapshot is already visible, VuGen hides it.

3 Use the expanded menu of **View** > **Snapshot** to view the recorded and/or replayed snapshots. You can also use the shortcut toolbar buttons to display the desired view:

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

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

### Troubleshooting Snapshots

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

Several protocols, such as Citrix and SAPGUI, allow you to disable the capturing of snapshots during recording using the Recording options.

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

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

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

- The script was recorded with VuGen version 6.02 or earlier.
- The imported actions do not contain snapshots.

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

### Snapshot Files

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

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

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

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

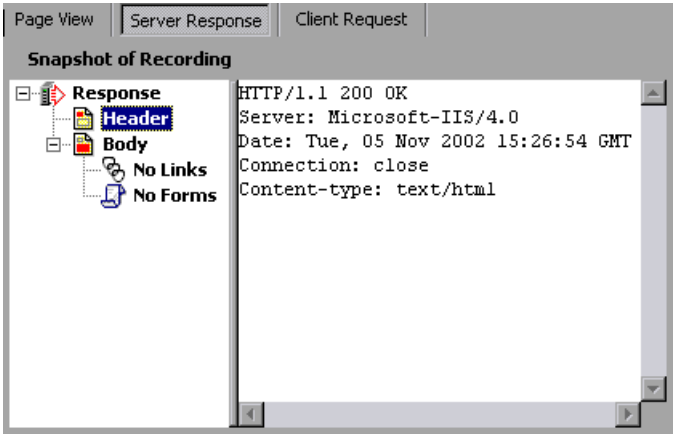
```
ctrx_sync_on_window("ICA Administrator Toolbar", ACTIVATE, 768, 0, 33,
        573, "snapshot12", CTRX_LAST);
```

### Web Vuser Snapshot Tabs

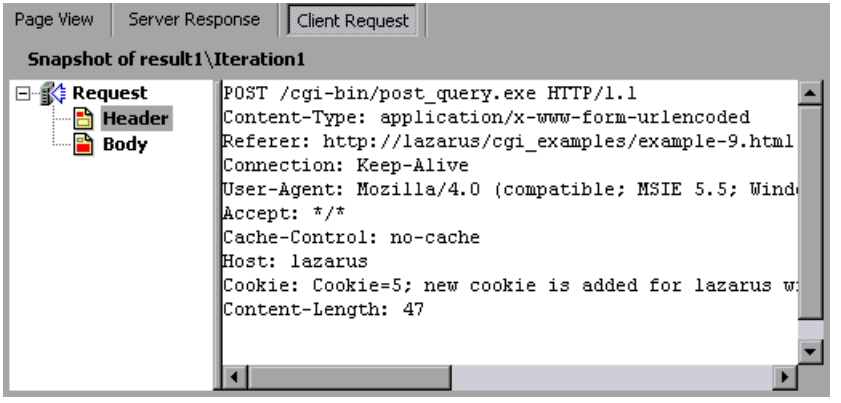
In the Snapshot window for Web Vusers, the following tabs are available:

**Page View:** Display the snapshot in HTML as it would appear in a browser. This button is available for both the recording and replay snapshots. Use this view to make sure you are viewing the correct snapshot. In this view, however, you do not see the values that need to be correlated.

**Server Response:** Displays the server response HTML code of the snapshot. This button is available for both the recorded and replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body with the title, links, forms, and so forth.




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



## Viewing Script Thumbnails

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

### Viewing Thumbnails in Tree View

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

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

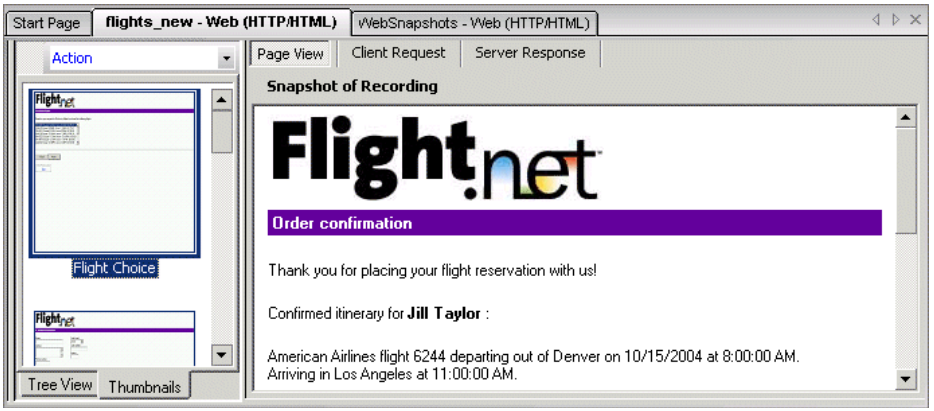
---

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

---

**To view the thumbnails from Tree View:**

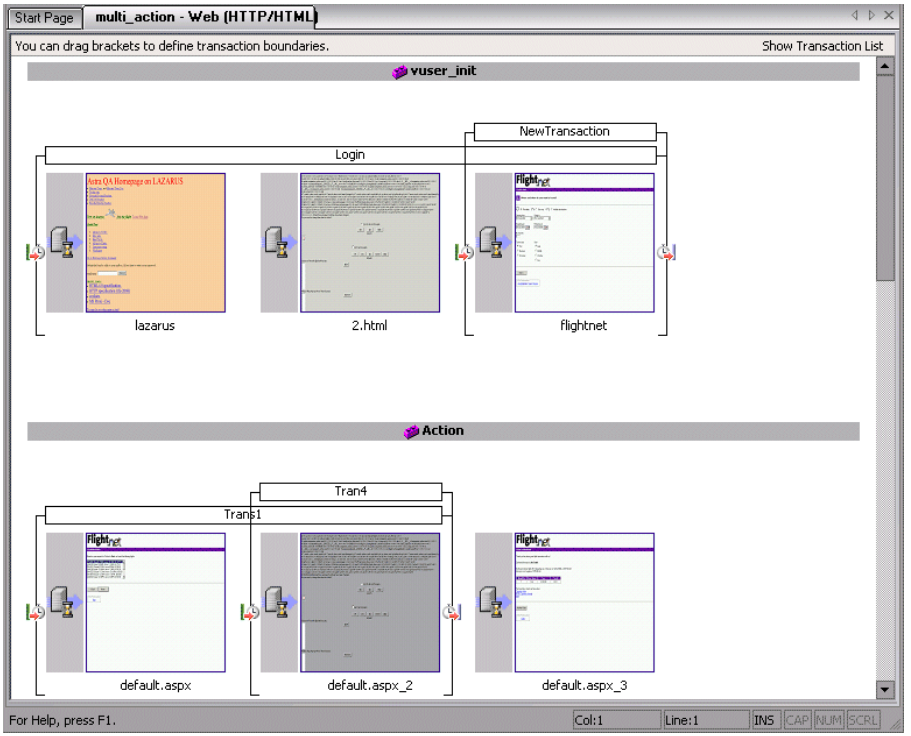
1. Click the **Thumbnail** tab at the bottom of the left pane.
2. Click the desired thumbnail image to open the thumbnail's snapshot in the right pane.



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

### Viewing Thumbnails in the Workflow Wizard

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




**To view thumbnails in the Transaction Editor:**

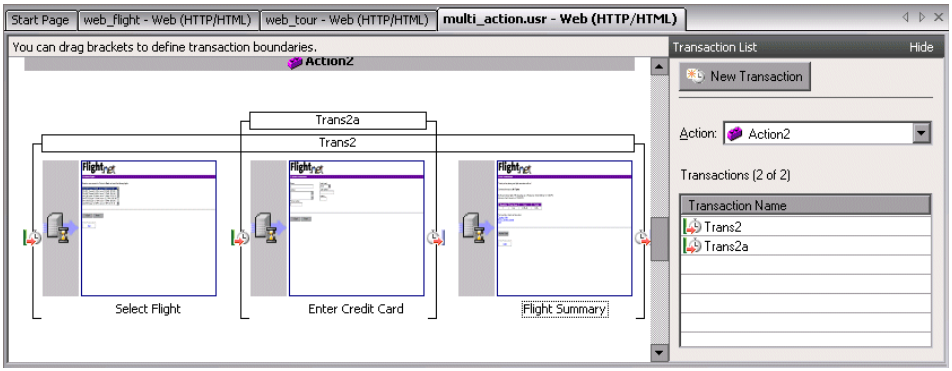
**1** Click the **Tasks** button on the toolbar to open the task list pane.

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

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

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

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




## Working with Thumbnails

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

### To view a thumbnail as a larger image

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

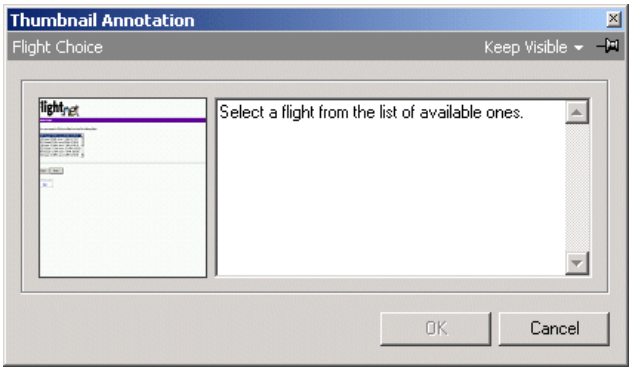
### To rename a snapshot:

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

**2** Type in the desired text.

**To annotate a snapshot:**

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

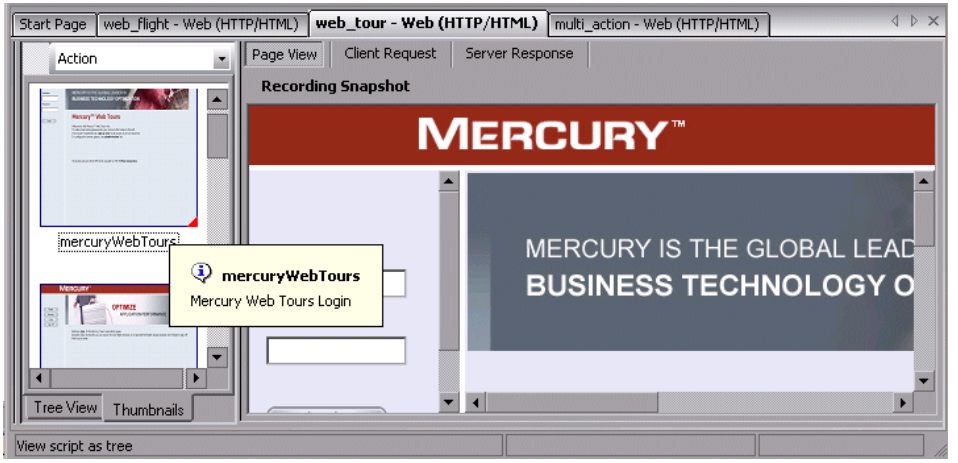



**2** Type text into the right pane of the Thumbnail Annotation dialog box.

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

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

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




## Running Vuser Scripts with VuGen

In order to perform testing or monitor an application with your Vuser script, you need to incorporate it into a LoadRunner scenario, Business Process Monitor profile, or Tuning Console session step. Before doing this, you check the script's functionality by running it from VuGen. For more information, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

If the script replay is successful, you can then integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

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 12, "Configuring Run-Time Settings."

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

Advanced users can compile a recorded script to create an executable program. For more information, see Chapter 7, "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 Mercury VuGen API. This API enables Vusers to communicate directly with a server. VuGen displays a list of all of the supported protocols when you create a new script. For syntax information about all of the Vuser functions, refer to the *Online Function Reference* (**Help** > **Function Reference**).

### General Vuser Functions

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

- Get run-time information about a Vuser, its Vuser Group, and its host.
- Add transactions and synchronization points to a Vuser script. For example, the **lr_start_transaction** (**lr.start_transaction** in Java) function marks the beginning of a transaction, and **the lr_end_transaction** (**lr.end_transaction** in Java) function marks the end of a transaction. See Chapter 7, "Enhancing Vuser Scripts" for more information.
- Send messages to the output, indicating an error or a warning.

See "Using C Vuser Functions" on page 32 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 5, "Setting Script Generation Preferences."

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

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

```
#include "as_web.h"


Action1()
{

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

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

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

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

      lr_start_transaction("Purchase_Order");
…
```

For more information about using C functions in your Vuser scripts, see Chapter 7, “Enhancing Vuser Scripts.” For more information about modifying a Java script, see Chapter 29, “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 7, "Enhancing Vuser Scripts."

The following list shows the general API functions for ANSI C scripts. This includes all protocols except for Java, VB, and GUI. For a list of the Java functions, seeChapter 29, "Programming Java Scripts." For a list of the VB functions, see the *Online Function Reference*, (**Help > Function Reference**).

**Transaction Functions:**

| | |
|---|---|
| **lr_end_sub_transaction** | Marks the end of a sub-transaction for performance analysis. |
| **lr_end_transaction** | Marks the end of a 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. |

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

## Command Line Parsing Functions

| | |
|---|---|
| **lr_get_attrib_double** | Retrieves a *double* type variable used on the script command line. |
| **lr_get_attrib_long** | Retrieves a *long* type variable used on the script command line. |
| **lr_get_attrib_string** | Retrieves a string used on the script command line. |

### Informational Functions

| | |
|---|---|
| **lr_user_data_point** | Records a user-defined data sample. |
| **lr_whoami** | Returns information about a Vuser to the Vuser script. Not applicable for Business Availability Center. |
| **lr_get_host_name** | Returns the name of the host executing the Vuser script. |
| **lr_get_master_host_name** | Returns the name of the machine running the LoadRunner Controller or Tuning Console. Not applicable for Business Availability Center. |

### String Functions

| | |
|---|---|
| **lr_eval_string** | Replaces 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. |
| **lr_save_datetime** | Saves the current date and time to a parameter. |
| **lr _advance_param** | Advances to the next available parameter. |
| **lr _decrypt** | Decrypts an encoded string. |
| **lr_eval_string_ext** | Retrieves a pointer to a buffer containing parameter data. |
| **lr_eval_string_ext_free** | Frees the pointer allocated by **lr_eval_string_ext**. |
| **lr_save_searched_string** | 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

| | |
|---|---|
| **lr_debug_message** | Sends a debug message to the Output window or the Business Process Monitor log files. |
| **lr_error_message** | Sends an error message to the Output window or the Business Process Monitor log files. |
| **lr_get_debug_message** | Retrieves the current message class. |
| **lr_log_message** | Sends a message to a log file. |
| **lr_output_message** | Sends a message to the Output window or the Business Process Monitor log files. |
| **lr_set_debug_message** | Sets a debug message class. |
| **lr_vuser_status_message** | Generates and prints formatted output to the Controller or Console Vuser status area. Not applicable for Business Availability Center. |
| **lr_message** | Sends a message to the Vuser log and Output window or the Business Process Monitor log files. |

## Run-Time Functions

| | |
|---|---|
| **lr_load_dll** | Loads an external DLL. |
| **lr_peek_events** | Indicates where a Vuser script can be paused. |
| **lr_think_time** | Pauses script execution to emulate think time—the time a real user pauses to think between actions. |
| **lr_continue_on_error** | Specifies an error handling method. |
| **lr_rendezvous** | Sets a rendezvous point in a Vuser script. Not applicable for Business Availability Center. |

# Getting Help on Functions

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

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

## Online Function Reference

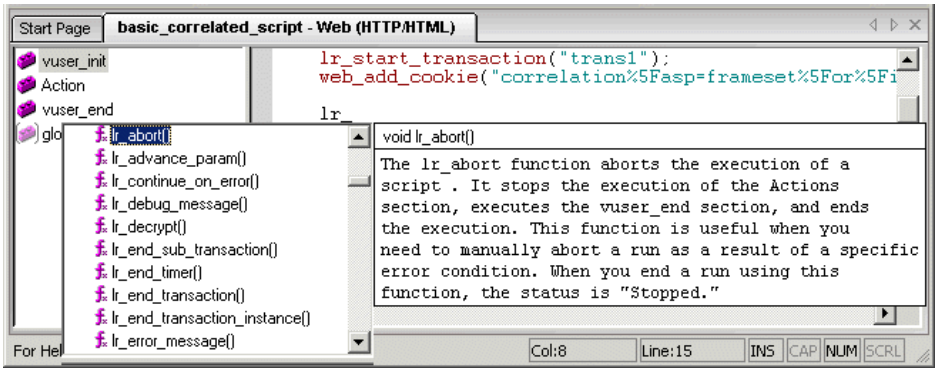
The *Online Function Reference* contains detailed syntax information about all of the VuGen functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

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

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

## Word Completion

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

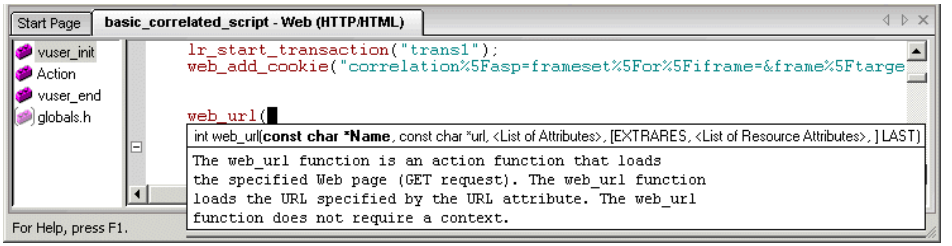



To use one of the displayed functions, select it, or scroll to the desired item and then select it. VuGen inserts the function at the location of the cursor. To close the list box, press the Esc key.

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

## Show Function Syntax

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




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

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

## Header File

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

In most cases, the name of the header file corresponds to the prefix of the protocol. For example, Database functions that begin with an **lrd** prefix, are listed in the **lrd.h** file.

The following table shows the header files that are associated with the most commonly used protocols:

| Protocol | File |
|---|---|
| Citrix | **ctrxfuncs.h** |
| COM/DCOM | **lrc.h** |
| Database | **lrd.h** |
| FTP | **mic_ftp.h** |
| General C function | **lrun.h** |
| IMAP | **mic_imap.h** |
| LDAP | **mic_mldap.h** |
| MAPI | **mic_mapi.h** |
| Oracle NCA | **orafuncs.h** |
| POP3 | **mic_pop3.h** |
| RealPlayer | **lreal.h** |
| SAPGUI | **as_sapgui.h** |
| Siebel | **lrdsiebel.h** |
| SMTP | **mic_smtp.h** |
| Terminal Emulator | **lrrte.h** |
| Tuxedo | **lrt.h** |
| WAP | **as_wap.h** |
| Web | **as_web.h** |
| Web Services | **wssoap.h** |
| Windows Sockets | **lrs.h** |

# 3

# Using the Workflow Wizard

VuGen's Workflow Wizard enables helps you develop a Vuser script that can be used for load testing or monitoring.

This chapter describes:

- About the Workflow Wizard
- Viewing the Task Pane
- Recording Steps
- Verifying the Script
- Enhancing the Script
- Prepare for Load
- Finishing Your Script

***The following information applies to all types of Vuser scripts.***

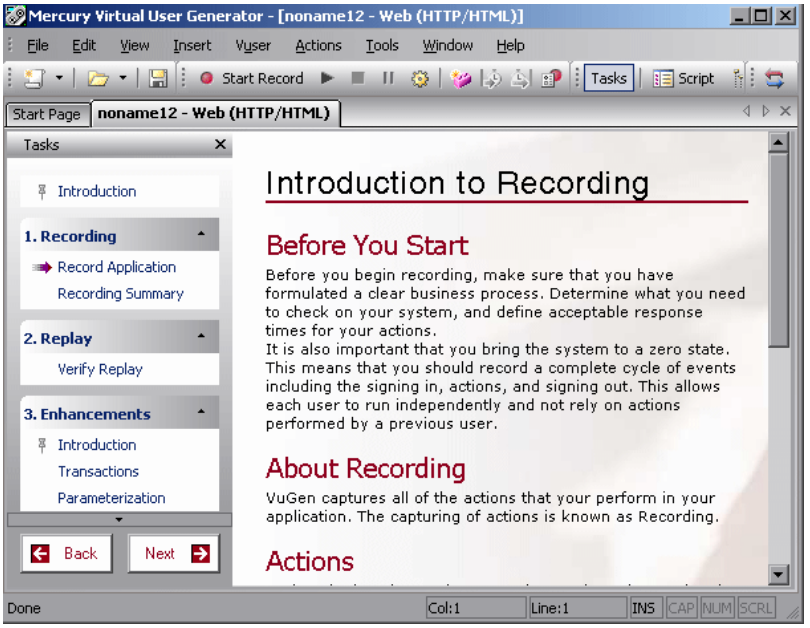
## About the Workflow Wizard

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

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

# Viewing the Task Pane

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




Note that the initial tasks differ slightly between Web, Web Services and non-recordable protocols, such as Custom C Vusers.

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

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

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

## Recording Steps

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

### Recording Application

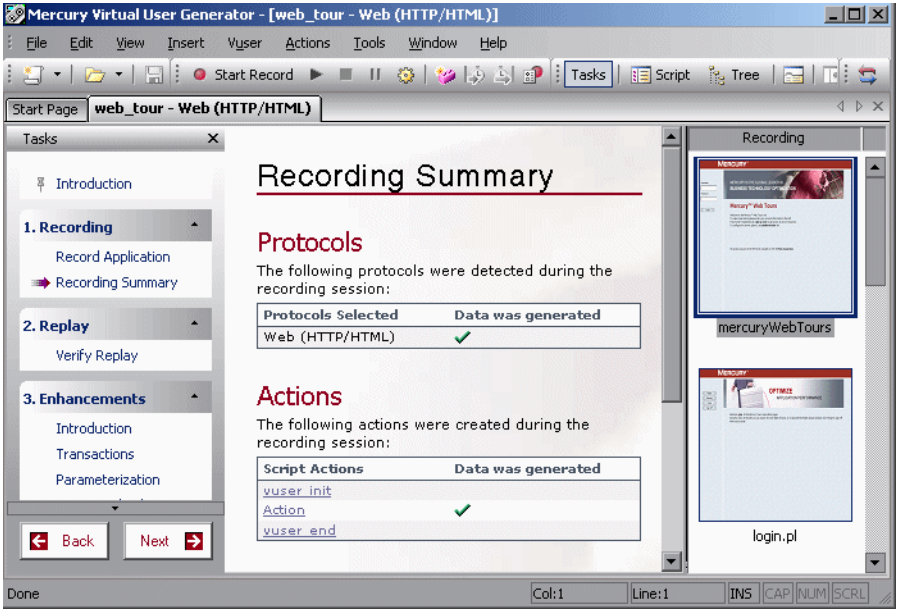
This wizard step provides an introduction to the recording process and contains the following sections.

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

### Recording Summary

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

This step also provides thumbnails of the recorded snapshots.




## Verifying the Script

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

### Verify Replay

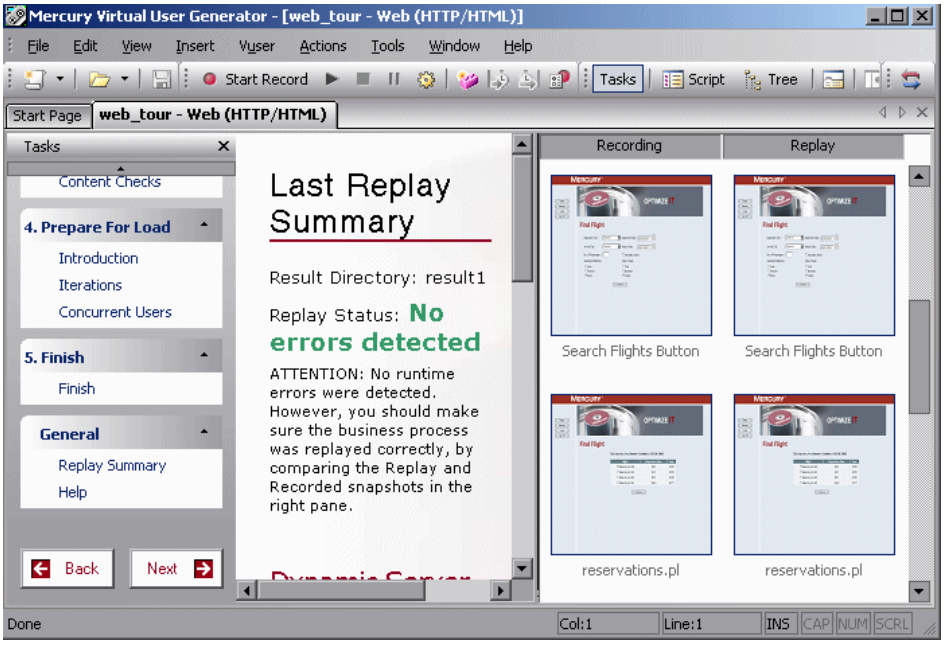
This wizard step provides an introduction to verification and contains the following sections.

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

### Replay Summary

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

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




---

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

---

# Enhancing the Script

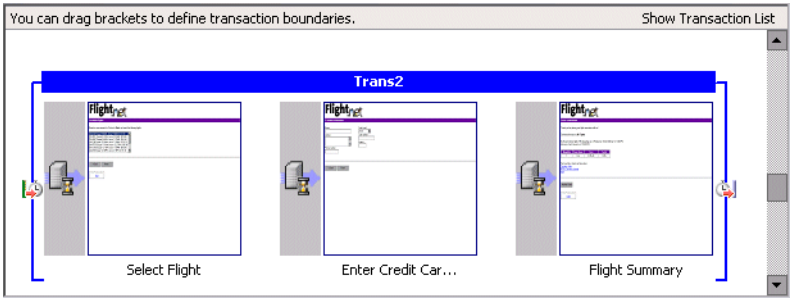
The Enhancement Section has three primary steps: Transactions, Parameterization, and Content Check.

## Transactions

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

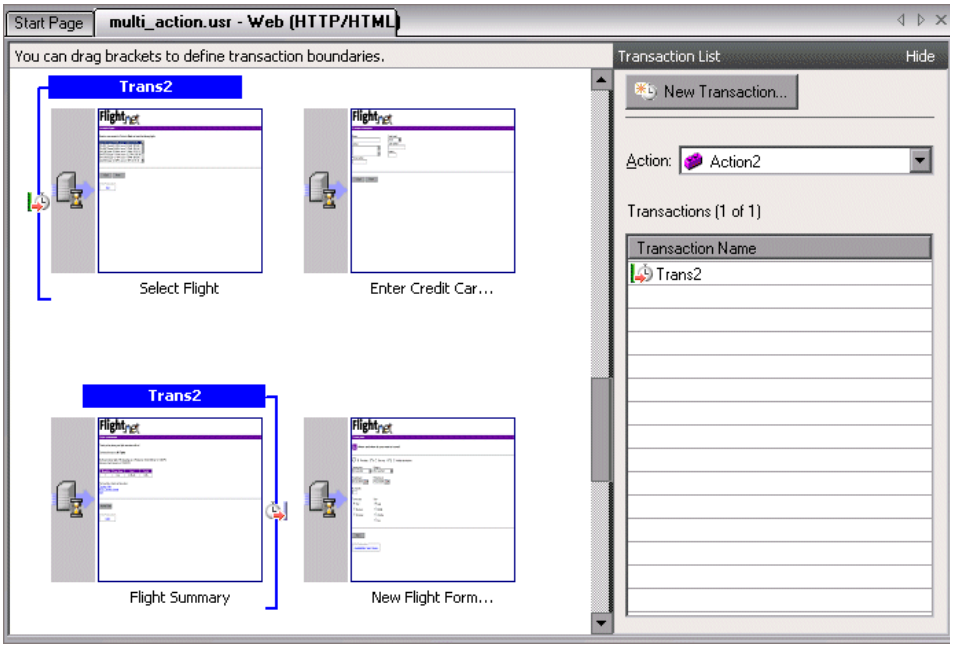
By default, VuGen only shows thumbnails for the primary steps in your script. To show thumbnails for all of the steps in your script, choose **View** > **Show All Thumbnails**.

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

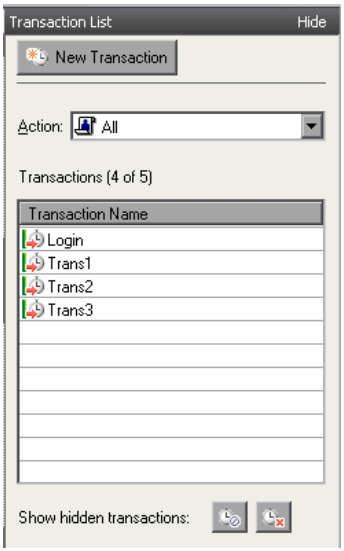


### Working with the Transactions List

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



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




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

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

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

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

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

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

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

### Defining Transactions in the Transaction Editor

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

**To define a transaction:**

1 Click **Transactions** in the Task list to open the Transaction Editor.

2 In the thumbnail area (middle pane), scroll down to the steps that you want to mark as a a transaction.

---

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

---

3 To mark a single step as a transaction, click on a thumbnail and choose **New Single-Step Transaction** from the right-click menu. VuGen prompts you to provide a name for the new transaction. If you want to expand the transactions at a later time, you can drag the transaction brackets to include additional steps.

4 To mark multiple steps as a transaction, click in the thumbnail area and choose **New Transaction** from the right-click menu or click the **New Transaction** button in the right pane.

5 VuGen shows instructions in the status area above the thumbnails:

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

Click the thumbnail of the transaction's first step.

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

Click the thumbnail of the transaction's last step.

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

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

To complete the transaction, press the Enter key.

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

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

**7** Repeat the above steps for additional transactions.

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

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

### Guidelines for the Transaction Editor

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

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

---

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

---

### Parameterization

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

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

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

### Content Check

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

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

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

# Prepare for Load

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

- Iterations
- Concurrent Users

### Iterations

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

**To set the number of iterations:**

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

### Concurrent Users

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

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

# Finishing Your Script

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

It contains the following sections:

- **Create a Scenario** - to run a load test on your system using the LoadRunner Controller.
- **Upload to Performance Center** - to run a test through a Performance Center server installation.
- **Upload to Quality Center** - to add a test to the test repository.

# 4

# Recording with VuGen

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

This chapter describes:

- About Recording with VuGen
- Vuser Script Sections
- Creating New Virtual User Scripts
- Adding and Removing Protocols
- Choosing a Virtual User Category
- Creating a New Script
- Opening an Existing Script
- Recording Your Application
- Ending and Saving a Recording Session
- Viewing the Recording Logs
- Using Zip Files
- Importing Actions
- Providing Authentication Information
- Regenerating a Vuser Script

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

# About Recording with VuGen

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

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

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

While recording, you can insert transactions, comments, and rendezvous points into the script. For details, see Chapter 7, "Enhancing Vuser Scripts."

# Vuser Script Sections

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

| Script Section | Used when recording... | Is executed when... |
|---|---|---|
| *vuser_init* | a login to a server | the Vuser is initialized (loaded) |
| *Actions* | client activity | the Vuser is in "Running" status |
| *vuser_end* | a logoff procedure | the Vuser finishes or is stopped |

When you run multiple iterations of a Vuser script, only the *Actions* sections of the script are repeated—the *vuser_init* and *vuser_end* sections are not repeated. For more information on the iteration settings, see Chapter 12, "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 29, "Programming Java Scripts."

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

---

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

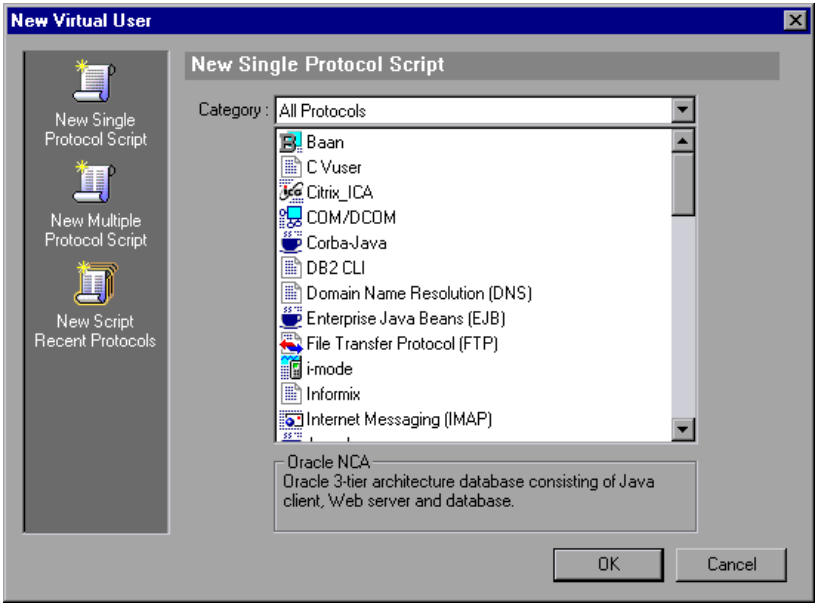
---

# Creating New Virtual User Scripts

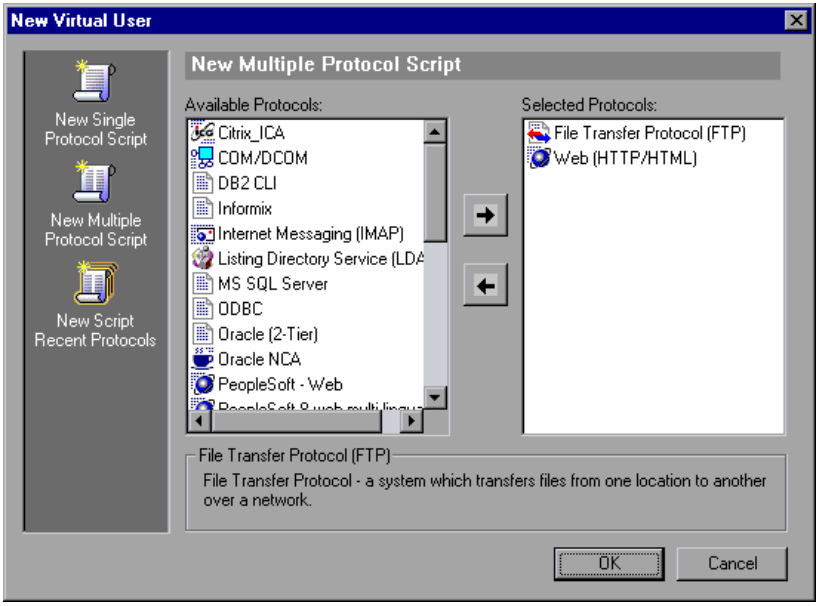
VuGen allows you to create new scripts by recording in either single or multi-protocol mode.

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

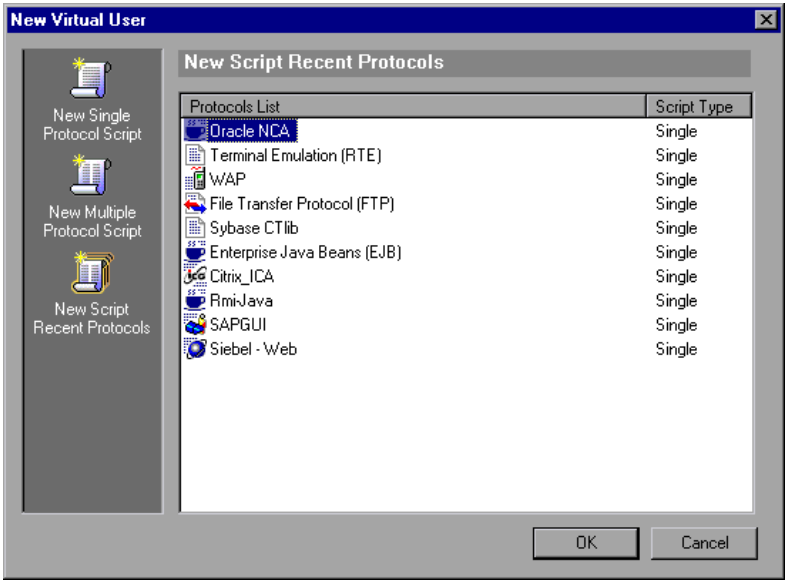
**New Single Protocol Script:** Creates a single protocol Vuser script. This is the default option when the Startup dialog box opens. To create a single protocol script, choose a category from the Category list (see "Choosing a Virtual User Category" on page 60), and select a protocol in the protocol list under that category.



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



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




When you record a single protocol, VuGen only records the specified protocol. When you record in multi-protocol mode, VuGen records the actions in several protocols. Multi-protocol scripts are supported for the following protocols: COM, FTP, IMAP, Oracle NCA, POP3, RealPlayer, Window Sockets (raw), SMTP, and Web. 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 VuGen supports a large variety of protocols, some of the recording steps that follow apply only to specific protocols.

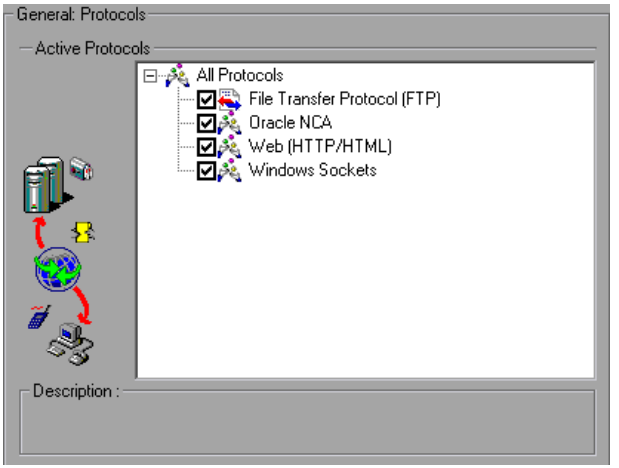
For all Java language Vusers (CORBA, RMI, Jacada, and EJB) see Chapter 17, "Recording Java Language Vuser Scripts" for details about recording, or the chapter discussing the specific protocol.

## Adding and Removing Protocols

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

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

Select the check boxes adjacent to the protocols you want to record in the next recording session. Clear the check boxes adjacent to the protocols you do not want to record in the next recording session.



## Choosing a Virtual User Category

The Vuser types are divided into the following categories:

- **All Protocols:** a list of all supported protocols in alphabetical order
- **Application Deployment Solution:** For the Citrix protocol
- **Client/Server:** For MS SQL, ODBC, Oracle 2-Tier, DB2 CLI, Sybase Ctlib, Sybase Dblib, Windows Sockets, and DNS 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, Microsoft .NET, Web (HTTP/HTML), Web Services, and the dual Web/Winsocket protocols
- **Enterprise Java Beans:** For EJB Testing and Rmi-Java protocols
- **ERP/CRM:** For Oracle NCA, Oracle Web Applications 11i, Peoplesoft Enterprise, Peoplesoft-Tuxedo, SAP-Web, SAPGUI, SAPGUI/SAP-Web dual, and Siebel (Siebel-DB2 CLI, Siebel-MSSQL, Siebel-Web, and Siebel-Oracle) protocols
- **Legacy:** For Terminal Emulation (RTE)
- **Mailing Services:** Internet Messaging (IMAP), MS Exchange (MAPI), POP3, and SMTP
- **Middleware:** Jacada and Tuxedo (6, 7) protocols
- **Streaming:** For Real and Media Player (MMS) protocols
- **Wireless:** For i-Mode, VoiceXML, and WAP protocols

# Creating a New Script

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

**To create a new Vuser script:**

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

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

---

**Note:** When recording certain Oracle NCA applications, you only need to choose **Oracle NCA**—not **Web Protocol**. For details, see Chapter 58, "Creating Oracle NCA Vuser Scripts."

---

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.

# Opening an Existing Script

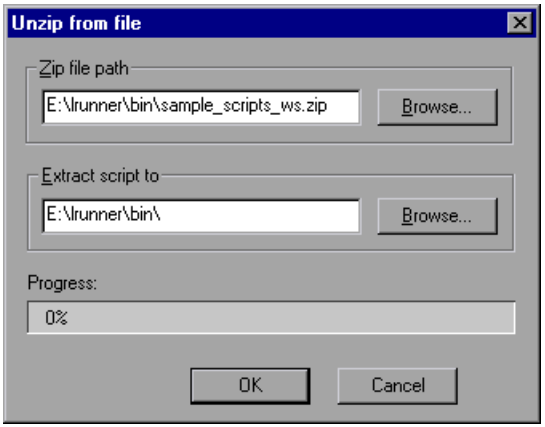
If you already have a script on the local machine or network, you can modify it and record additional actions.

**To open an existing script:**

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

2 To open a file from a Quality Center repository (LoadRunner only), see "Opening Scripts from a Quality Center Project" on page 229.

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




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

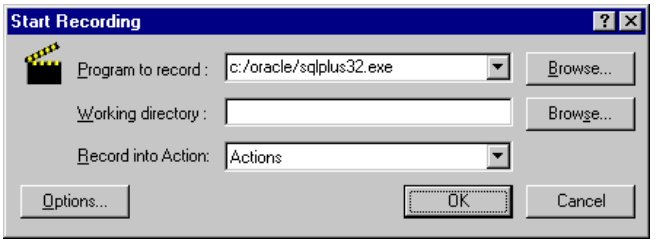
# Recording Your Application

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

**To begin recording:**

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

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




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

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

**4** For Internet Applications, fill in the relevant information:

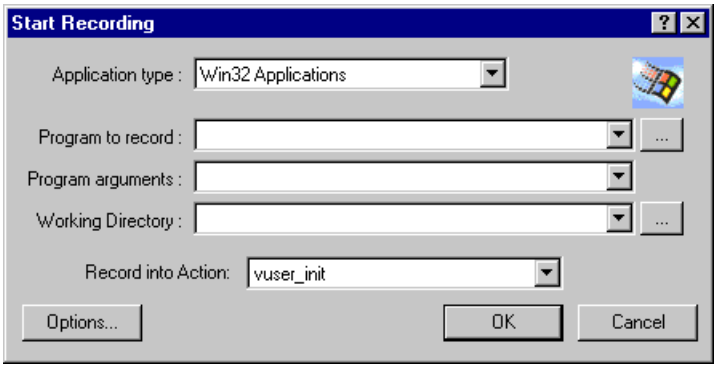



**Program to record:** Select the browser or Internet application to record.

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

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

**5** For Win32 Applications, fill in the relevant information:




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

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

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

**6** In the **Record into Action** box, select the section into which you want to record. Initially, the available sections are *vuser_init*, *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.




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

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

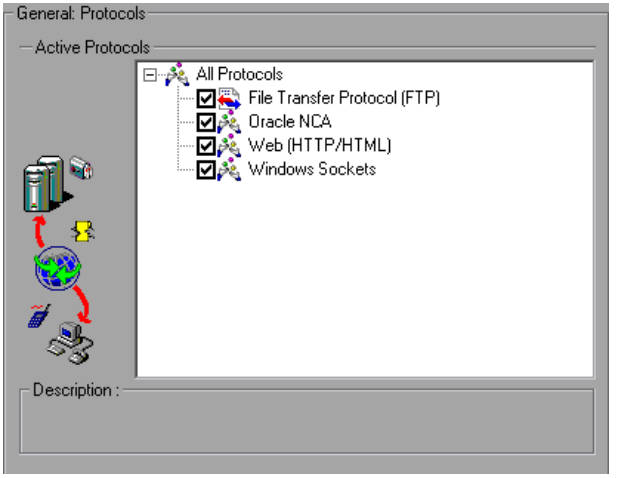


**8** Click **Options** or the **Recording Options** button to open the Recording options dialog box and set the recording options. The available options may vary, depending on the recorded protocol. For more information, see their respective chapters.

**9** To choose a language for code generation and to set the scripting options, click the **Script** tab. For details, see Chapter 5, “Setting Script Generation Preferences.”

**10** To specify port information, click the **Port Mapping** tab. This is useful when recording SSL applications on a non-standard port. Review the list of ports. If the port you are using is not on the list, you can specify the information using the Port Mapping options. For more information, see Chapter 6, “Configuring the Port Mappings.”

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




You are now ready to begin recording.

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

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

**14** VuGen starts your application and the Recording toolbar opens.




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

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

# Ending and Saving a Recording Session

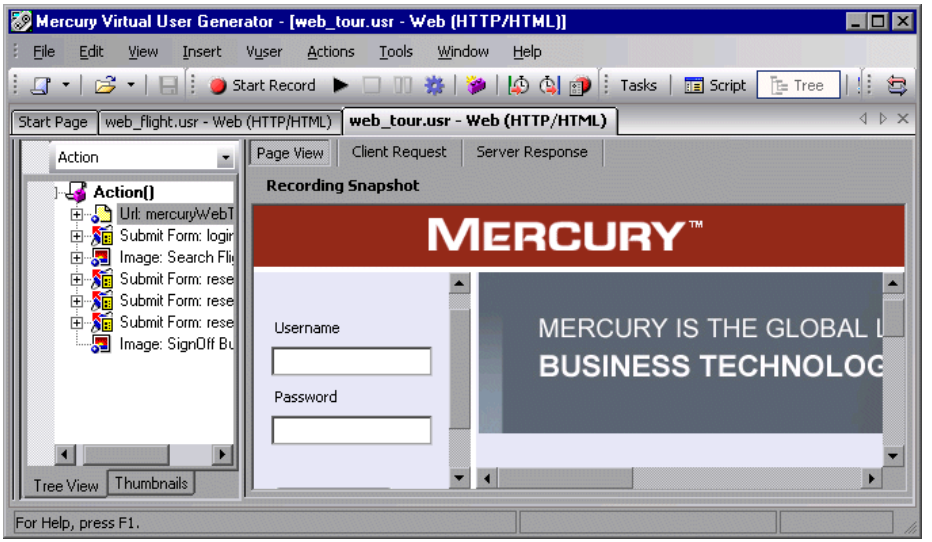
After you record a typical business process, you complete the recording session by performing the closing steps of your business process and saving the Vuser script.

**To complete the recording:**

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



**2** Click the **Stop Recording** button on the Recording toolbar. The VuGen editor displays all the recorded steps, (or the recorded functions if you began in script view).






**3** Click **Save** to save the recorded session. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name. **Note:** Do not name the script *init*, *run* or *end*, since these names are used by VuGen.

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

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

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

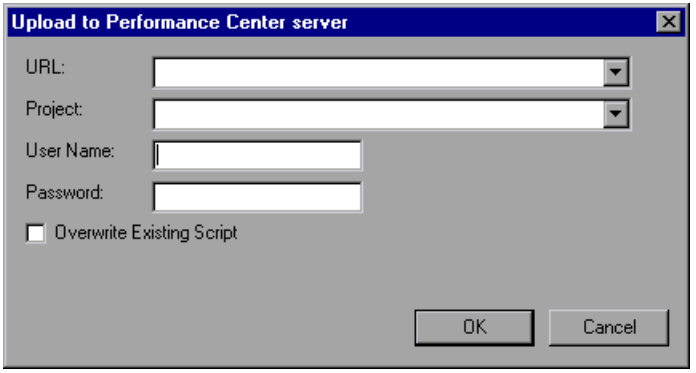
Click **OK**.

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

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

Enter an email address and send your email.

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




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

# Viewing the Recording Logs

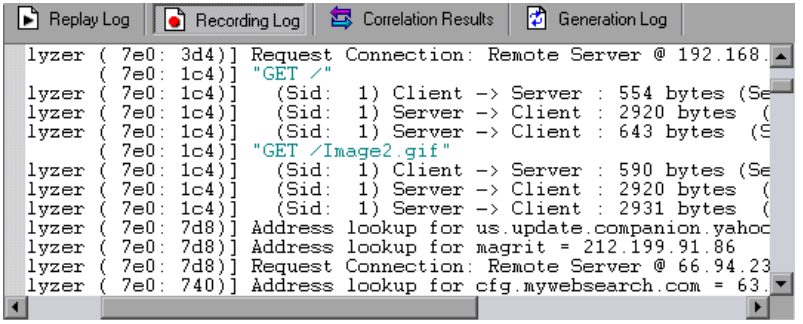
After recording, you can view the contents of the *vuser_init*, *Actions*, and *vuser_end* sections in the VuGen script editor. To display an action, select the action name in the left pane.

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

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

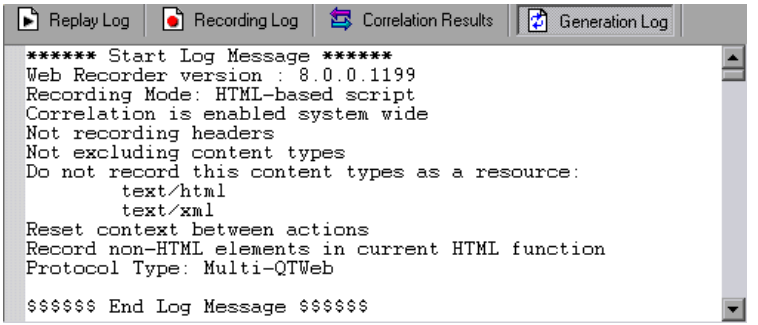
## Recording Log

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



### Generation Log

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




## Using Zip Files

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

### Importing from a Zip file

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

### Working from a Zip file

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

### Exporting to a Zip File

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

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

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

### Zip and Email

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

## Importing Actions

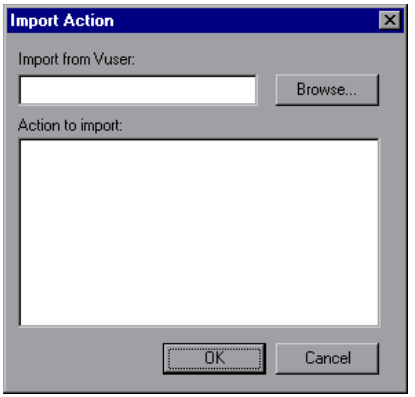
For Vuser types that support multiple actions, you can import actions into your script from another Vuser script. You can only import actions from Vusers of the same type. Note that any parameters associated with the imported action, will be merged with the script. The available options are:

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

**Action to Import:** Select the Action you want to import.

**To import actions into the current script:**

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




2 Click **Browse** to select a Vuser script. A list of the script's actions appears in the **Actions to Import** section.

3 Highlight an action and click **OK**. The action appears in your script.

4 To rearrange the order of actions, you must first enable action reordering. Right-click on any action and select **Enable Action Reorder**. Then drag the actions to the desired order. Note that when you reorder actions in the left pane of VuGen, it does not affect the order in which they are executed. To change the order of execution, use the Pacing node of the Run-Time settings as described in Chapter 12, "Configuring Run-Time Settings."

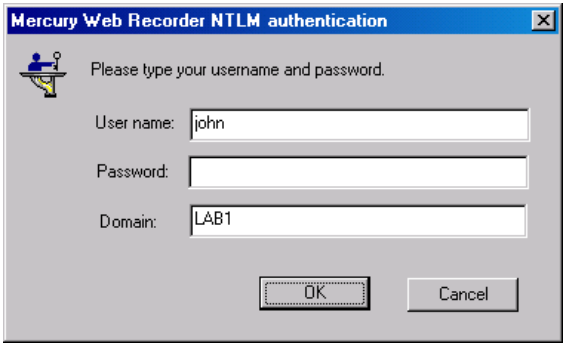
## Providing Authentication Information

The following section only applies to multi-protocol recording.

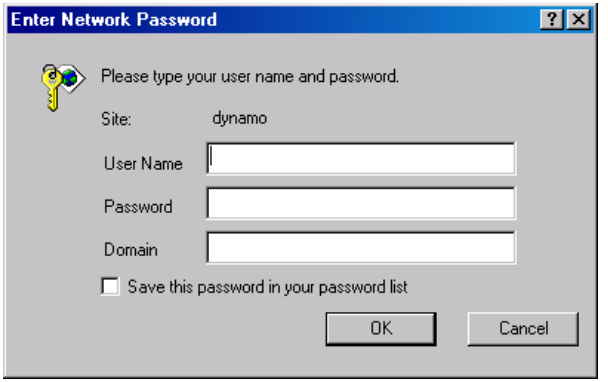
When recording a Web session that uses NTLM authentication, your server may require you to enter details such as a user name and password.

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

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




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



### Generating a web_set_user function

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

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

```
web_set_user("domain1\\dashwood",
        lr_decrypt("4042e3e7c8bbbcfde0f737f91f"),
       "sussex:8080");
```

➤ If you cancel the Mercury Web Recorder NTLM Authentication dialog box without entering information, VuGen generates a **web_set_user** function for you to edit manually.

```
web_set_user("domain1\\dashwood,
        "Enter NTLM Password Here",
        "sussex:8080");
```

---

**Note:** If you enter a password manually, it will appear in the script as is, presenting a security issue. To encrypt the password, select it and choose **Encrypt string** from the right-click menu. VuGen encrypts the string and generates an **lr_decrypt** function, used to decode the password during replay. For more information about encrypting strings, see "Encrypting Text" on page 109.

---

## Regenerating a Vuser Script

After recording a script, you can enhance it by adding transactions, rendezvous, messages, or comments. For more information, see Chapter 7, "Enhancing Vuser Scripts."

In addition, you can parameterize the script and correlate variables. For more information, see Chapter 8, "Working with VuGen Parameters."

If you need to revert back to your originally recorded script, you can regenerate it. This feature is ideal for debugging, or fixing a corrupted script. When you regenerate a script, it removes all of the manually added enhancements to the recorded actions. 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. Note that regeneration, only cleans up the recorded actions, but not those that were manually added.

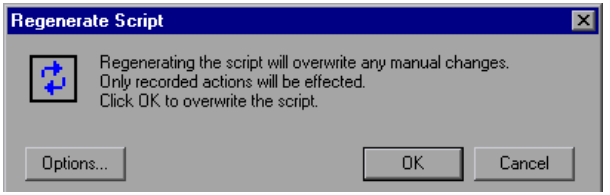
The following buttons are available from the Regenerate Script dialog box.

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

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

**To regenerate a multi-protocol Vuser script:**

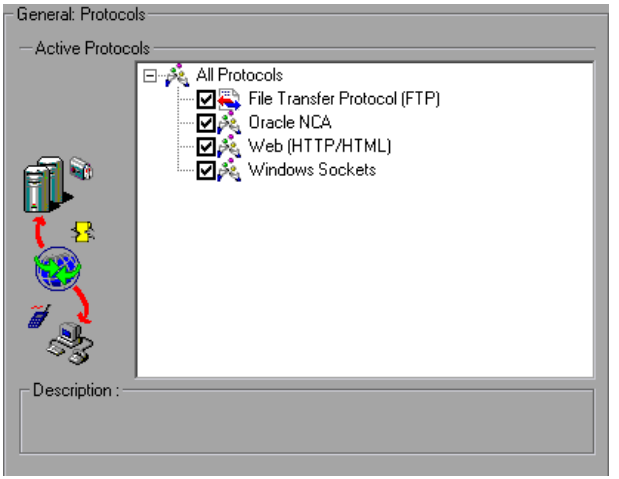
**1** Choose **Tools** > **Regenerate Script**. VuGen issues a warning indicating that all manual changes will be overwritten.



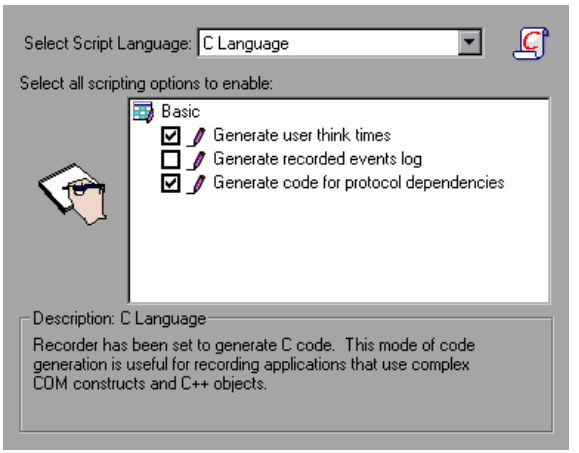

**2** Click **Options** to open the Regenerate Options dialog box.

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

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




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



# 5

# Setting Script Generation Preferences

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.

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

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

## About Setting Script Generation Preferences

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.

**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 all scripts, see "Applying the Basic Options" on page 78. To set the correlation options for non-C scripts, see "Understanding the Correlation Options" on page 80.

For further instructions, see "Setting Script Recording Options" on page 81.

## 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.

**Close all AUT processes when recording stops:** Automatically closes all of the application under test's (AUT) processes when VuGen stops recording (disabled by default).

**Explicit variant declaration** - Declare variant types explicitly in order to handle *ByRef* variants (Visual Basic for Applications only, disabled by default).

**Generate fixed think time after end transaction:** Add a fixed think time, in seconds, after the end of each transaction. When you enable this option, you can specify a value for the think time. The default is 3 seconds (disabled by default).

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

**Generate think time greater than threshold:** Use a threshold value for think time. If the recorded think time is less than the threshold, VuGen does not generate a think time statement. You also specify the threshold value. The default values is 3—if the think time is less than 3 seconds, VuGen does not generate think time statements. If you disable this option, VuGen will not generate any think times (enabled by default).

**Maximum number of lines in action file** - Create a new file if the number of lines in the action exceeds the specified threshold. The default threshold is 60000 lines (C Language only).

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

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

**Replace long strings with parameter** - Save strings exceeding the maximum length to a parameter. This option has an initial maximum length of 100 characters. The parameters and the complete strings are stored in the *lr_strings.h* file in the script's folder in the following format:

```
const char <paramName_uniqueID> =”string”.
```

This option allows you to have a more readable script. It does not effect the performance of the script (enabled by default).

**Track processes created as COM local servers:** Track the activity of the recorded application if one of its sub-processes was created as a COM local server (enabled by default).

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

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

For further instructions, see "Setting Script Recording Options" on page 81.

## 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, and so on (disabled by default).

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

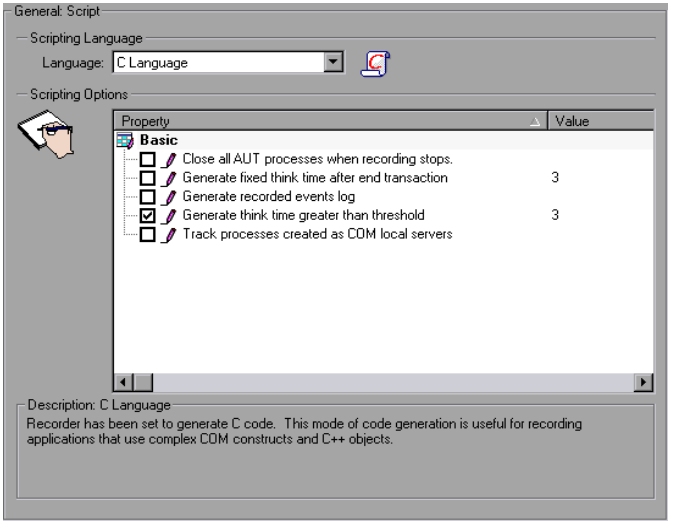
For further instructions, see "Setting Script Recording Options" on page 81.

# Setting Script Recording Options

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

**To set the script recording options:**

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




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

# 6

# 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:

- About Configuring the Port Mappings
- Defining Port Mappings
- Adding a New Server Entry
- Setting the Auto-Detection Options
- 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.

## Defining Port Mappings

VuGen uses the Port Mapping settings to direct traffic via a specific server:port combination to the desired communication protocol.

**Network-level server address mappings for:** Specifies the mappings per protocol. For example, to show only the FTP mappings, choose FTP.

**New Entry:** Opens the Server Entry dialog box, allowing you to add a new mapping. See “Adding a New Server Entry” on page 86.

**Edit Entry:** Opens the Server Entry dialog box, allowing you to edit the selected entry.

**Delete Entry:** Deletes the selected entry.

**Options:** Opens the Advanced Settings dialog box to enable auto-detection of the communication protocol and SSL level. See “Setting the Auto-Detection Options” on page 88.

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

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

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

In addition, the following guidelines apply:

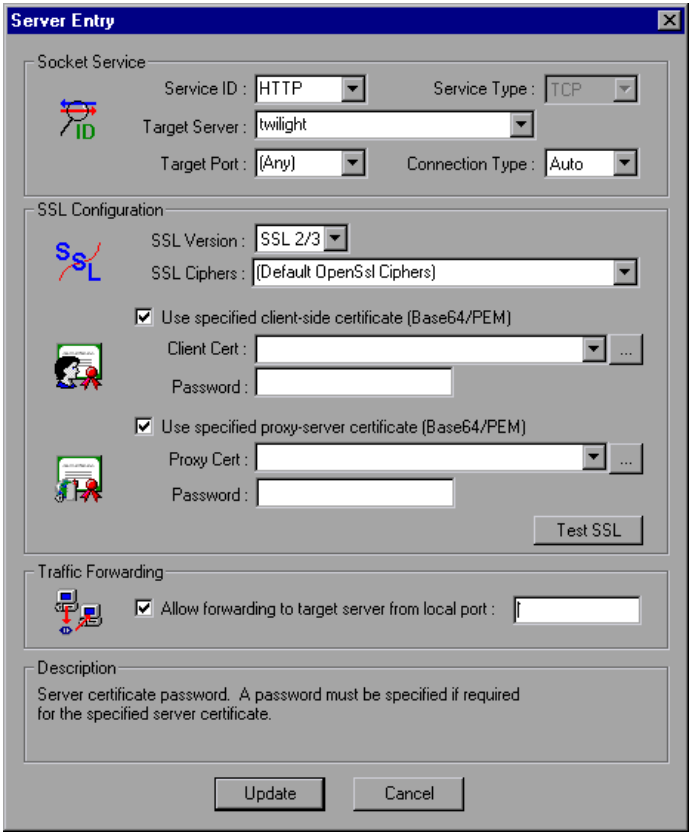
- **Port 0:** Port number 0 indicates any port.
- **Forced mapping:** If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol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

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

  For further instructions, see "Setting the Port Mapping Recording Options" on page 90.

# Adding a New Server Entry

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




### Socket Service

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

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

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

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

**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.

### SSL Configuration

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

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

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

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

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

### Traffic Forwarding

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

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

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

For further instructions, see "Setting the Port Mapping Recording Options" on page 90.

## 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.

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

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

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

**Update:** Accepts the auto-detection options and closes the dialog box.

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

- The protocol signature closely resembles an existing protocol, resulting in erroneous processing.
- There is no unique signature for the protocol.
- The protocol uses SSL encryption, and therefore cannot be recognized on a WinSock level.

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

For further instructions, see “Setting the Port Mapping Recording Options” on page 90.

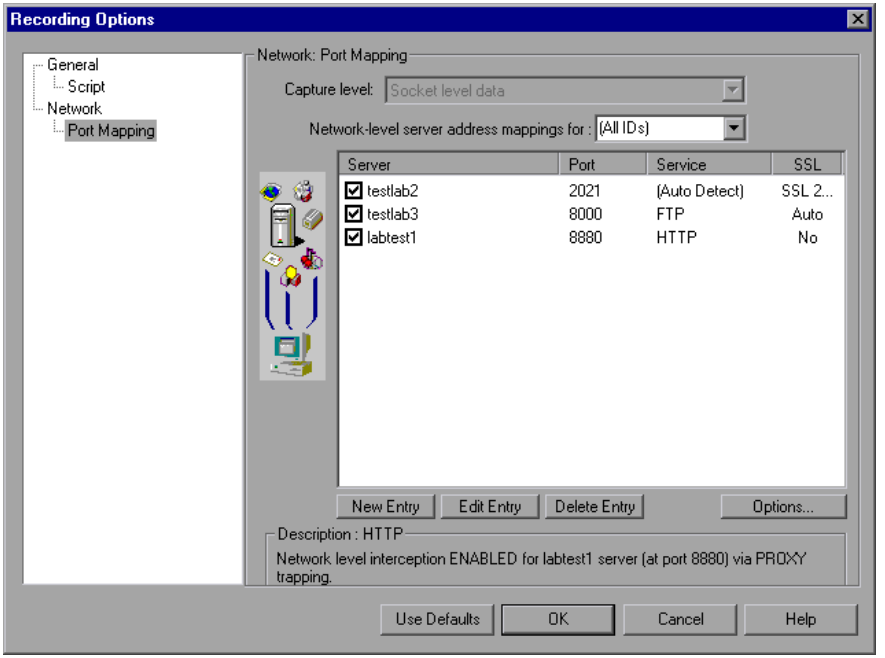
## Setting the Port Mapping Recording Options

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

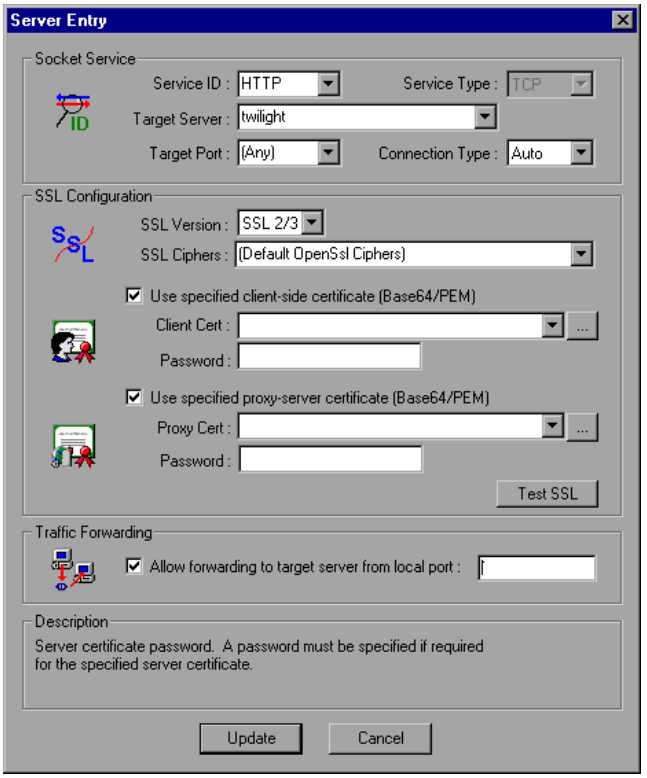
- The toolbar button: 
- The keyboard shortcut: Ctrl+F7
- The Tools menu: choose **Tools** > **Recording Options**

**To set the port mapping recording options:**

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



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




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

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

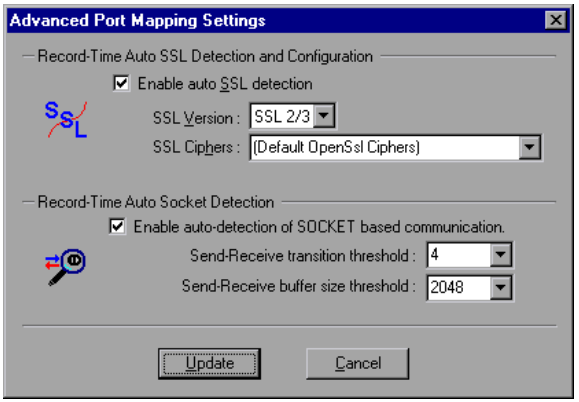
Specify the **SSL Version**, **SSL Cipher**. To use a certificate, select **Use specified client-side certificate** or **Use specified proxy-server certificate** and specify the user information.

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

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

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

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




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

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

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

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

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

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

11 Click **OK**.

# 7

# 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:

- About Enhancing Vuser Scripts
- Inserting Transactions into a Vuser Script
- Inserting Rendezvous Points (LoadRunner and Tuning only)
- 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
- Encrypting Text
- Encoding Passwords Manually
- Adding Files to the Script Folder

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

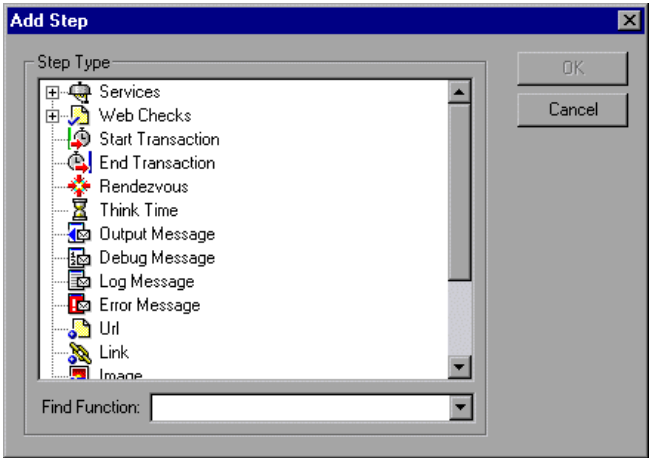
# About Enhancing Vuser Scripts

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

**To add a new step to your script.**

1 Place the cursor at the desired location.

2 Choose **Insert** > **New Step**. The Add Step dialog box opens with the relevant steps for the current protocol.




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

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

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

## General Vuser Functions

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

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

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

## Protocol-Specific Vuser Functions

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

### Standard ANSI C Functions

You can enhance your Vuser scripts by adding standard ANSI C functions. ANSI C functions allow you to add comments, control flow statements, conditional statements, and so forth to your Vuser scripts. You can add standard ANSI C functions to any type of Vuser script. For details, see "Guidelines for Using C Functions" on page 421.

## Inserting Transactions into a Vuser Script

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

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

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

You can create transactions either during or after recording. To add transactions after recording, use the Transaction editor to graphically mark the steps of a transaction, as described in "Transactions" on page 46. Alternatively, use the **Insert** menu to add **Start Transaction** and **End Transaction** markers.

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

## Marking the Beginning of a Transaction

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

**To mark the start of a transaction:**



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




**2** Type a transaction name in the Transaction Name box. Transaction names must begin with a letter or number and may contain letters, numbers, or the following characters !, $, %, &, ', -, [, ^, _, `, <, >, {, }, |, or ~. Do not use the period (.) character.

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

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

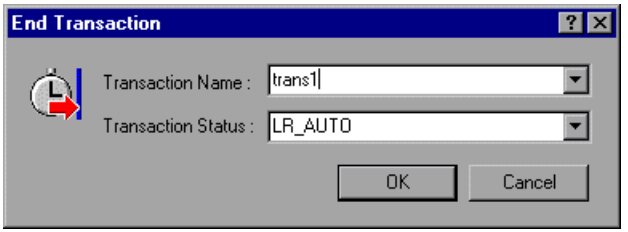
## Marking the End of a Transaction

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

**To mark the end of a transaction:**



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




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

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

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

---

**Note:** You can create *nested* transactions—transactions within transactions. If you nest transactions, close the inner transactions before closing the outer ones—otherwise the transactions won't be analyzed properly.

---

# Inserting Rendezvous Points (LoadRunner and Tuning only)

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

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

---

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

---

**To insert a rendezvous point:**



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




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

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

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

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

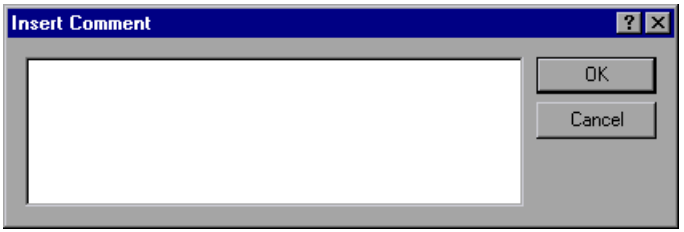
## Inserting Comments into a Vuser Script

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

**To insert a comment:**



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




2 Type the comment into the text box.

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

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

---

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

---

## Obtaining Vuser Information

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

| | |
|---|---|
| **lr_get_attrib_string** | Returns a command line parameter string. |
| **lr_get_host_name** | Returns the name of the machine running the Vuser script. |
| **lr_get_master_host_name** | Returns the name of the machine running the Controller or Tuning Console. Not applicable for Administration Console. |
| **lr_whoami** | Returns the name of a Vuser executing the script. Not applicable for Administration Console. |

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

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

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

## Sending Messages to Output

Using the Message type functions in your Vuser script, you can send customized error and notification messages to the output and log files. For example, you could insert a message that displays the current state of the client application. The LoadRunner Controller and Tuning Console display these messages in the Output window. You can also save these messages to a file.

When working with Business Availability Center, you can use Message type functions to send error and notification messages to the Web site or Business Process Monitor log files. For example, you could insert a message that displays the current state of the Web-based application.

---

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

---

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

| | |
|---|---|
| **lr_debug_message** | Sends a debug message to the Output window or the Business Process Monitor log file. |
| **lr_error_message** | Sends an error message to the Output window or the Business Process Monitor log files. |
| **lr_get_debug_message** | Retrieves the current message class. |
| **lr_log_message** | Sends an output message directly to the log file, *output.txt*, located in the Vuser script directory. This function is useful in preventing output messages from interfering with TCP/IP traffic. |
| **lr_output_message** | Sends a message to the Output window or the Business Process Monitor log files. |
| **lr_set_debug_message** | Sets a message class for output messages. |
| **lr_vuser_status_message** | Sends a message to the Vuser status area in the Controller or Tuning Console. Not applicable for Administration Console. |
| **lr_message** | Sends a message to the Vuser log and Output window or the Business Process Monitor log files. |

---

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

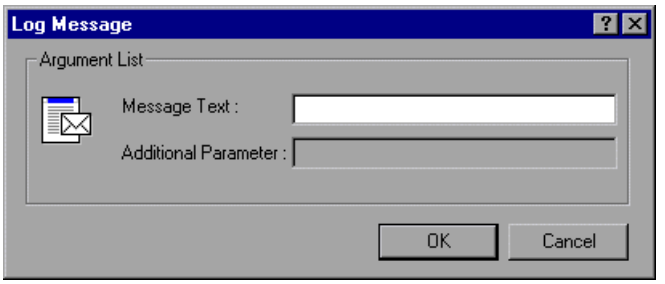
---

## Log Messages

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

**To insert an lr_log_message function:**

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




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

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

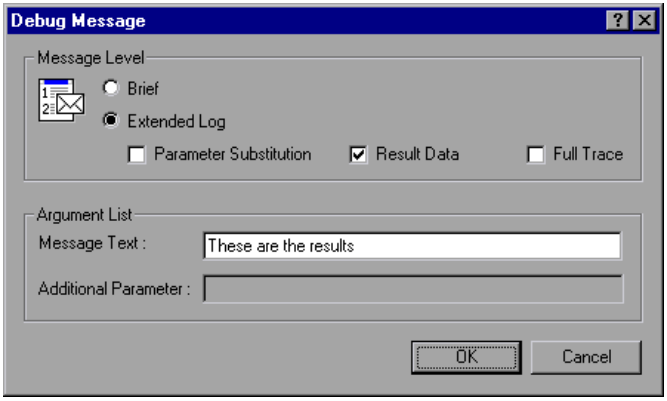
## Debug Messages

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

**To insert a debug function:**

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

2 Select the **Debug Message** step and click **OK**. The Debug Message dialog box opens.




3 Select a message level, **Brief** or **Extended Log**. If you choose Extended Log, indicate the type of information to log: **Parameter Substitution**, **Result Data**, or **Full Trace**.

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

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

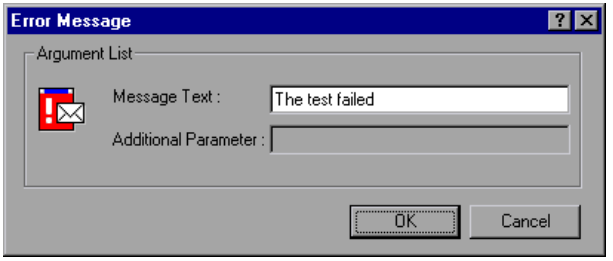
## Error and Output Messages

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

**To insert an error or output message function:**

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

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




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

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

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

## Handling Errors in Vuser Scripts During Execution

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

- Using run-time settings. You can specify the **Continue on Error** run-time setting. The **Continue on Error** run-time setting applies to the entire Vuser script. You can use the **lr_continue_on_error** function to override the **Continue on Error** run-time setting for a portion of a script. For details, see "Error Handling" on page 193.
- 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 65, "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 your record a Vuser script, VuGen records the actual think times and inserts appropriate **lr_think_time** statements into the Vuser script. You can edit the recorded **lr_think_time** statements, and manually add more **lr_think_time** statements to a Vuser script.

**To manually add a think time statement:**

1. Place the cursor at the desired location.
2. Choose **Insert** > **Add Step**. The Add Step dialog box opens.

**3** Select **Think Time** and click **OK**. The Think Time dialog box opens.




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

---

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

---

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

## Handling Command Line Arguments

You can pass values to a Vuser script at run-time by specifying command line arguments when you run the script. When using the Tuning Console, you can specify the command line arguments within the Run-Time settings dialog box. For more information, see "Configuring Additional Attributes Run-Time Settings" on page 191.

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

| | |
|---|---|
| **lr_get_attrib_double** | Retrieves double precision floating point type arguments |
| **lr_get_attrib_long** | Retrieves long integer type arguments |
| **lr_get_attrib_string** | Retrieves character strings |

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

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

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

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

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

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

## Encrypting Text

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

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

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

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

**To encrypt a string:**

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

**To restore an encrypted string:**

1 For protocols that have tree views, view the script in script view. Choose **View** > **Script View**.

2 Select the string you want to restore.

3 Select **Restore encrypted string (***string***)** from the right-click menu.

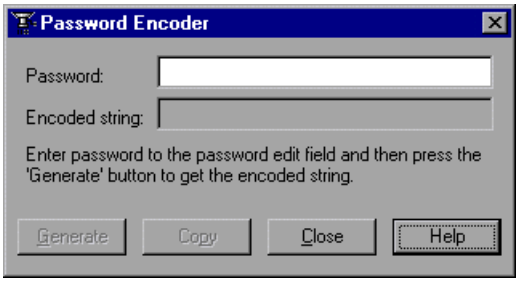
For more information on the **lr_decrypt** function, refer to the *Online Function Reference* (**Help** > **Function Reference**).

## Encoding Passwords Manually

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

**To encode a password:**

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




2 Enter the password in the **Password** box.

3 Click **Generate**. The Password Encoder encrypts the password and displays it in the **Encoded String** box.

4 Use the **Copy** button to copy and paste the encoded value into the Data Table.

5 Repeat the process for each password you want to encode.

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

## Adding Files to the Script Folder

You can add files to your script directory to make them available when running the script.

**To add a file:**

1 Choose **File** > **Add Files to Script...** while viewing the script.

2 Browse for the files and click **Open**. VuGen adds the selected files.

# 8

# Working with VuGen Parameters

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

This chapter describes:

- About VuGen Parameters
- Understanding Parameter Limitations
- Creating Parameters
- Understanding Parameter Types
- Defining Parameter Properties
- Using Existing Parameters
- Using the Parameter List
- Setting Parameterization Options

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

## About VuGen Parameters

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

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

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

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

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

The resulting Vusers then substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables. For more information about data sources, see "Understanding Parameter Types" on page 119.

Parameterizing a Vuser script has two advantages:

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

Parameterization involves the following two tasks:

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

## Understanding Parameter Limitations

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

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

```
lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long mli-
TextLen, LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int miDBErrorSe-
verity);
```

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

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

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

You could parameterize the recorded function to look like this:

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

---

**Note:** You can use the **lr_eval_string** function to "parameterize" a function argument that you cannot parameterize by using standard parameterization. In addition, you can use the **lr_eval_string** function to "parameterize" any string in a Vuser script.

For VB, COM, and Microsoft .NET protocols, you must use the **lr.eval string** function to define a parameter. For example, lr.eval_string( "{**Custom_param**}" ).

For more information on the **lr_eval_string** function, refer to the *Online Function Reference*.

---

## Creating Parameters

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

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

If you are in Script view:

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

---

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

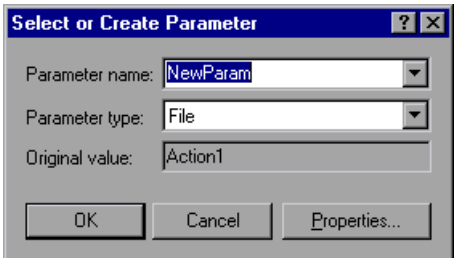
---

If you are in Tree view:

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

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

The Select or Create Parameter dialog box opens.




**Step 2: Give the parameter a name.**

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

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

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

---

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

---

**Step 3: Select a parameter type.**

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

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

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

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

For more detailed information about the different parameter types, see "Understanding Parameter Types" on page 119.

**Step 4: Define properties for the parameter type.**

1 Click **Properties**. The Parameter Properties dialog box opens.

2 Click **Create Table**. A message box opens. Click **OK**.

VuGen creates a table with one cell containing the argument's original value.

3 To add another value to the table, click **Add Row**, and enter the value.

Repeat this step to add more values to the table.

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

For more information, see "Defining Parameter Properties" on page 122.

**Step 5: Replace the argument with the parameter.**

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

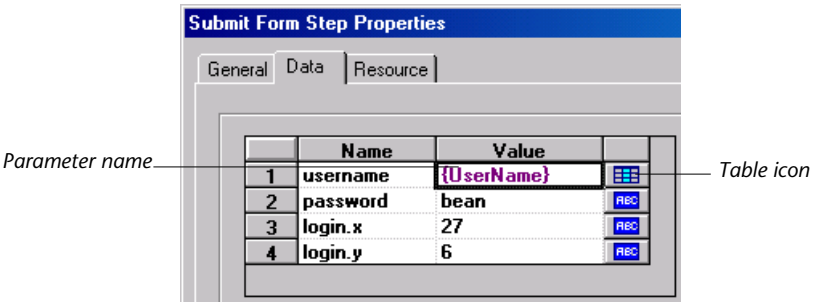
VuGen replaces the selected string in your script with the name of the parameter, surrounded by curly or round brackets.

**Note:** The default parameter braces are either curly or angle brackets, depending on the protocol type. You can change the parameter braces from the Parameterization tab in the General Options dialog box (select **Tools** > **General Options**). For more information, see "Setting Parameterization Options" on page 129.



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

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




## Understanding Parameter Types

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

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

### File or Table Parameter Types

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

### Data Files

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

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

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

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

---

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

---

### Data Tables

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

For information about defining data file or data table parameter properties, see Chapter 9, “File and Table Type Parameters.”

### Internal Data Parameter Types

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

For information about defining Internal Data parameter properties see "Setting Properties for Internal Data Parameter Types" on page 150.

### User-Defined Function Parameters

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

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

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

The arguments sent to this function are both NULL.

When you create the library, it is recommended that you use the default dynamic library path. That way, you do not have to enter a full path name for the library, but rather, just the library name. The Mercury 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;}
```

For information about defining User-Defined Function properties, see "Setting Properties for User-Defined Functions" on page 160.

# Defining Parameter Properties

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

**To define parameter properties in the Parameter Properties dialog box:**

**1 Open the Parameter Properties dialog box.**

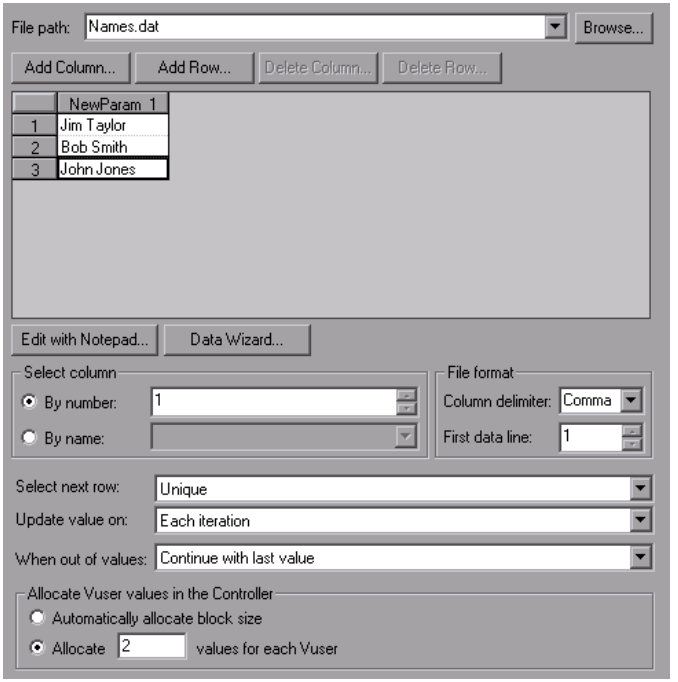
You open the Parameter Properties dialog box in one of the following ways:

- When you create a new Parameter as described in "Creating Parameters" on page 116, you click **Properties** in the Select or Create Parameter dialog box to open the Parameter Properties dialog box.
- In Script view, select the parameter, and choose **Parameter Properties** from the right-click menu.
- In Tree view, right-click the step containing the parameter whose properties you want to define, and select **Properties**. The Step Properties dialog box for the selected step opens.



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

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




**2 Define the parameter properties.**

- To define properties for File and Table type parameters, see Chapter 9, "File and Table Type Parameters."
- To define properties for internal data parameter types, see "Setting Properties for Internal Data Parameter Types" on page 150.
- To define properties for user-defined functions, see "User-Defined Function Parameters" on page 121.

**3 Close the Parameter Properties dialog box.**

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

**To define parameter properties in the Parameter List dialog box:**



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

For more information, see "Using the Parameter List" on page 127.

# Using Existing Parameters

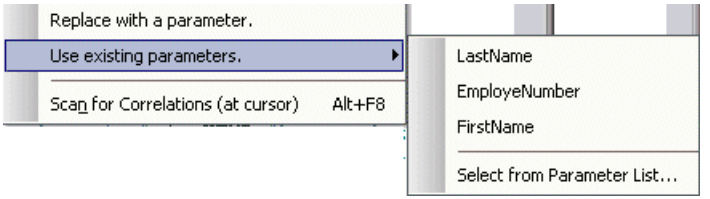
When you create a parameter, VuGen stores it in a parameter list. You can use an existing parameter to replace an argument, or to replace multiple occurrences of an argument.

## Replacing Strings Using Pre-defined Parameters

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

**To replace a string with a pre-defined parameter:**

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




3. Use one of the following options to select a parameter:
   - Select a parameter from the submenu list.
   - Choose **Select from Parameter List** to open the Parameter List dialog box, and select a parameter from the left pane.

     Using the **Parameter List** is convenient when you want to replace an argument with a previously defined parameter and, at the same time, view or modify that parameter's properties. For details on using the Parameter List, see "Using the Parameter List" on page 127.

## Replacing Multiple Occurrences

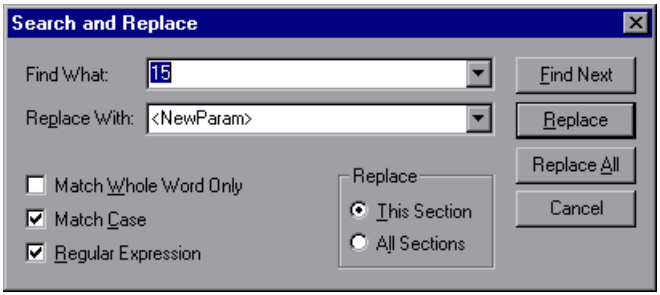
When you create a parameter, the system remembers the original value of the argument. You can use the **Search and Replace** function to replace selected or all occurrences of the same argument with the same parameter or another existing parameter.

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

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

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

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




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

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

---

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

---

### Restoring Original Strings

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

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

➤ In Script view, right-click on the parameter and select **Restore original value**.

➤ In Tree view:

➤ Right-click on the step that contains the parameter and click **Properties**.



➤ Click the table icon next to the parameter that you want to restore to its original value, and select **Undo Parameter**.

The original argument is restored.

# Using the Parameter List

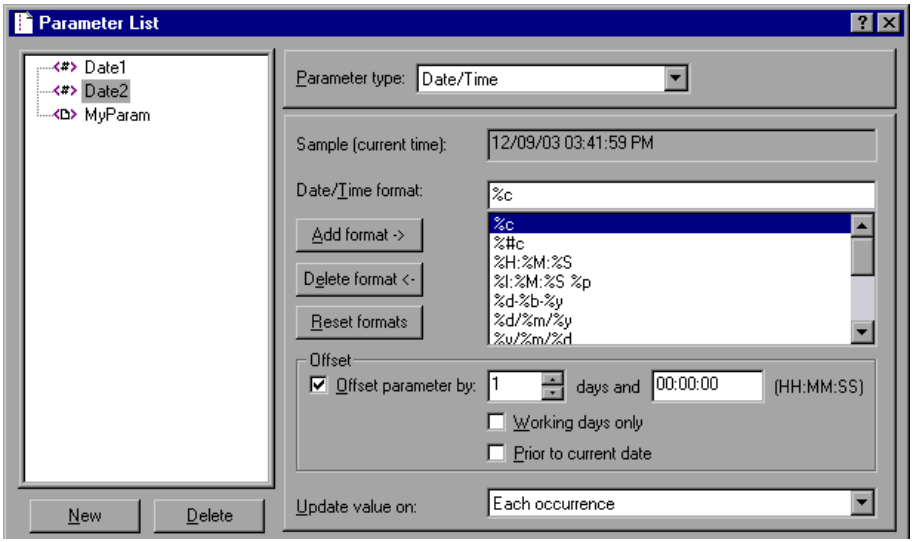
You use the Parameter List to examine all of the parameters, create new parameters, delete parameters, or modify a parameter properties.

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



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

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




**To modify a parameter's properties:**

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

For more information on setting parameter properties, see Chapter 10, "Setting Parameter Properties," and Chapter 9, "File and Table Type Parameters."

**To create a new parameter:**

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

---

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

---

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

---

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

---

**To delete an existing parameter:**

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

# Setting Parameterization Options

You set the parameterization options in the Parameterization tab of VuGen's General Options window.

**To set the Parametrization options:**

1. From the **Tools** menu, select **General Options**.
2. Click the **Parameterization** tab.
3. Set the style for the parameter braces as described below.
4. Set the global directory as described on page 130.
5. Click **OK** to close to the General Options window.

### Parameter Braces

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

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

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

---

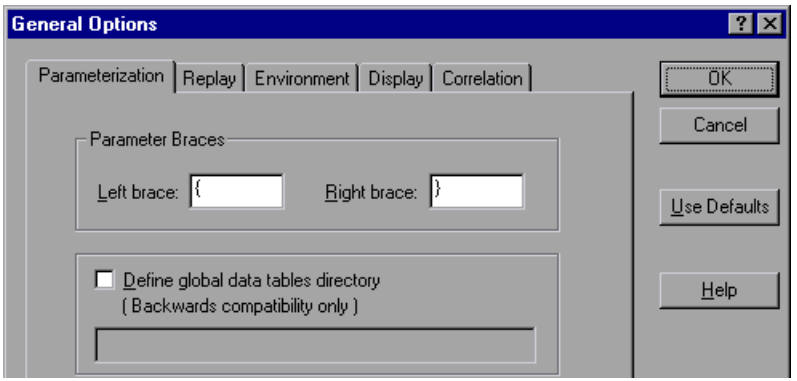
**Note:** The default parameter braces are angle or curly brackets, depending on the protocol type.

---

**To change the parameter brace style:**

1 Select **Tools** > **General Options** in VuGen. The General Options dialog box opens.

2 Select the **Parameterization** tab and enter the desired brace.




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

## Global Directory

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

In newer versions of VuGen, you specify the location of the data table either in the Parameter Properties dialog box or in the Parameter List dialog box. VuGen is able to retrieve the data from any location that you specify, be it the default script directory or another directory on the network. For more information, see "Data Files" on page 120.

**To set the global directory:**

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

# 9

# File and Table Type Parameters

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

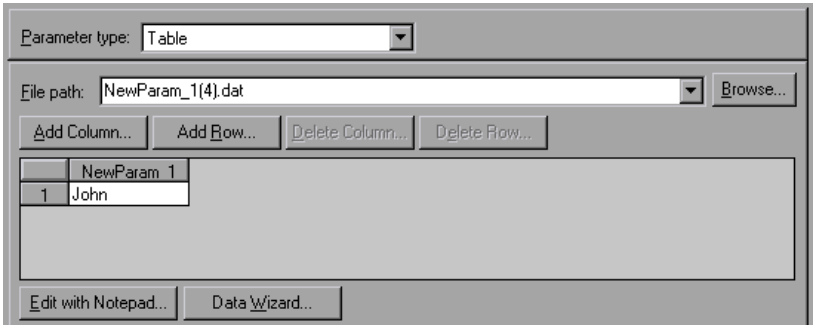
This chapter describes:

- Selecting or Creating Data Files or Data Tables
- Setting Properties for File Type Parameters
- Setting Properties for Table Type Parameters
- Choosing Data Assignment Methods for File/Table Type Parameters

# Selecting or Creating Data Files or Data Tables

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

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




**To select a source file or table for your data:**

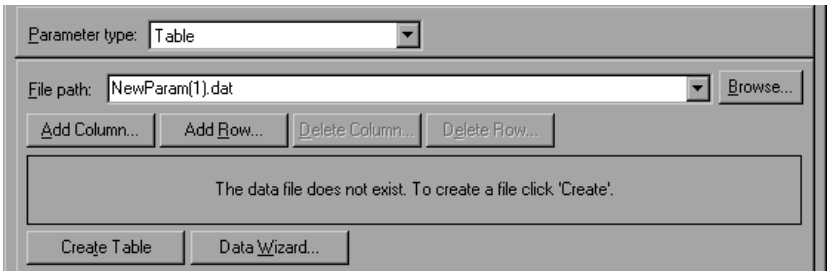
**1 Open the Parameter Properties dialog box or the Parameter List.**

For instructions, see "Defining Parameter Properties" on page 122.

**2 Select a table or create a new one.**

- If there are no tables (**.dat** files) listed in the file path list, or you want to create a new table, click **Create Table**. VuGen creates a new table with one cell, displaying the original value of the argument in the first column of the table.
- To open an existing data file, type the name of the **.dat** file in the **File path** box or choose a name from the drop-down list.

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




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

---

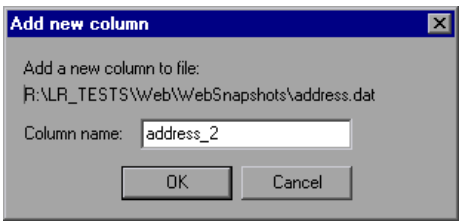
**Note:** You can also specify a global directory. Global directories are provided only for backward compatibility with earlier versions of VuGen. For more information, see "Global Directory" on page 130.

---

➤ To import data from an existing database, click **Data Wizard** and follow the wizard's instructions. For more information, see "Importing Data from an Existing Databases" on page 136.

**3 Add columns and rows to the table.**

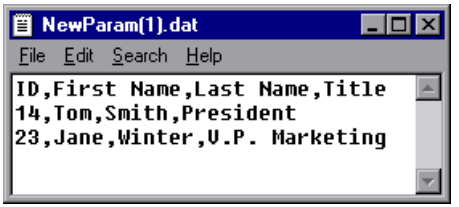
➤ To add additional columns to the table, choose **Add Column**. The Add new column dialog box opens. Enter a column name and click **OK.**




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

**4 Edit the data file.**

- Click within any cell to enter a value.
- To edit the data file from within Notepad, click **Edit with Notepad**. Notepad opens with the parameter's name in the first row and its original value in the second row. Enter additional column names and values into the file using a delimiter such as a comma or a tab to indicate a column break. Begin a new line for each table row (for each new row of data).



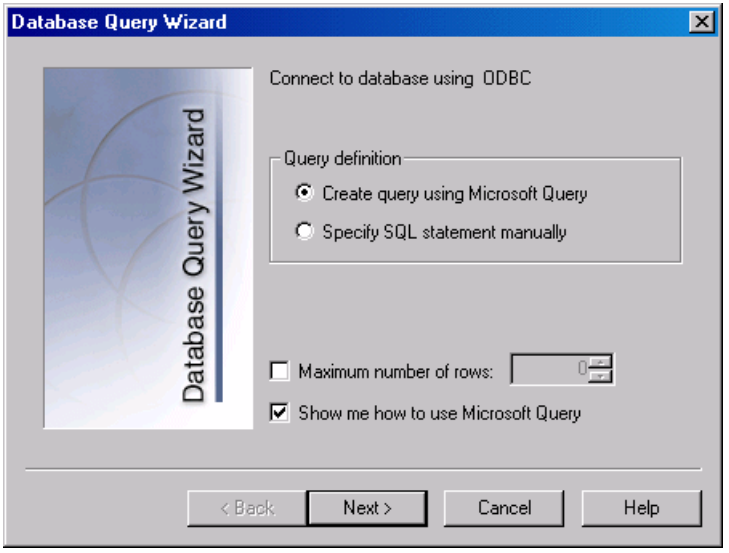

### Importing Data from an Existing Databases

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

- Creating a New Query
- Specifying an SQL Statement

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

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



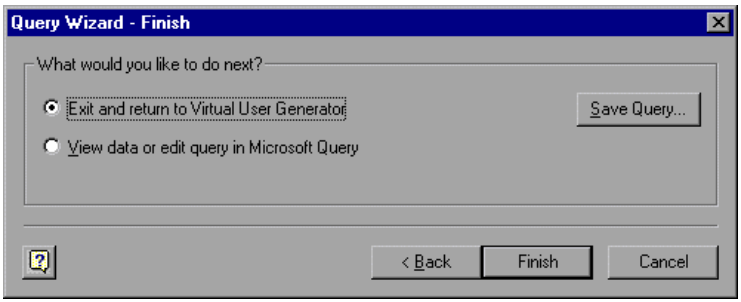

### Creating a New Query

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

**To create a new query:**

1. Select **Create query using Microsoft Query**. If you need instructions on Microsoft Query, select **Show me how to use Microsoft Query**.
2. Click **Finish**. If Microsoft Query is not installed on your machine, VuGen issues a message indicating that it is not available. Install MS Query from Microsoft Office before proceeding.
3. Follow the instructions in the wizard, importing the desired tables and columns.

4 When you finish importing the data, choose **Exit and return to Mercury Virtual User Generator** and click **Finish**. The database records appear in the Parameter Properties box as a data file.




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

5 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 140.

### Specifying an SQL Statement

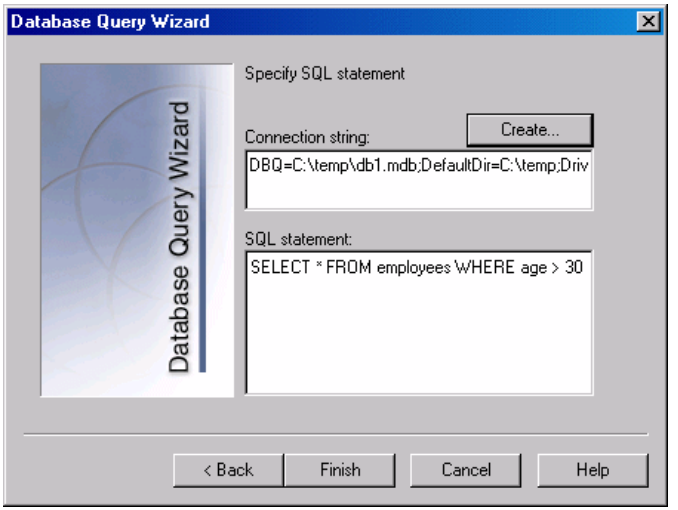
**To specify a database connection and SQL statement:**

1 Select **Specify SQL Statement**. Click **Next**.

2 Click **Create** to specify a new connection string. The Select Data Source window opens.

3 Select a data source, or click **New** to create a new one. The wizard guides you through the procedure for creating an ODBC data source. When you are finished, the connection string appears in the **Connection String** box.

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




**5** Click **Finish** to process the SQL statement and import the data. The database records appears in the Parameter Properties box as a data file.

**6** Set the data assignment properties. See “Setting Properties for File Type Parameters” on page 140.

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

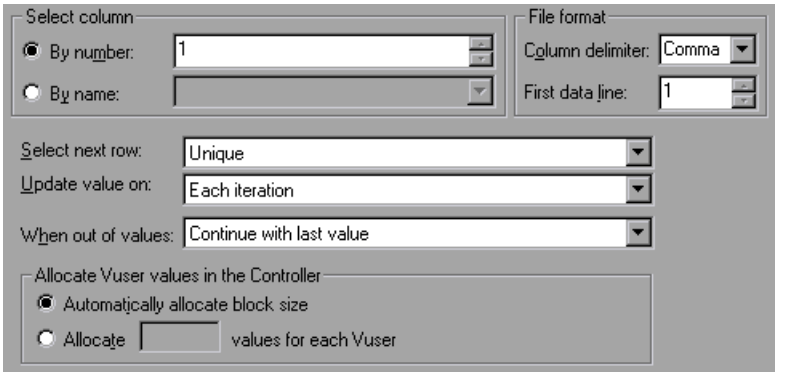
---

**Note:** You can also set the properties for a parameter from the Parameter List dialog box. In the left pane, select the parameter and then specify its properties in the right pane. See “Using the Parameter List” on page 127.

---

# Setting Properties for File Type Parameters

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




**To set the File type parameter properties:**

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

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

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

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

**3** In the **First data line** box of the **File format** section, select the first line of data to be used during Vuser script execution. The header is line 0. To begin with the first line after the header, specify 1. If there is no header, specify 0.

**4** Select a Data Assignment method from the **Select next row** list to instruct the Vuser how to select the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Type Parameters" on page 144.

**5** Choose an update option from the **Update value on** list. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information, see "Data Assignment and Update Methods for File/Table Parameters" on page 146.

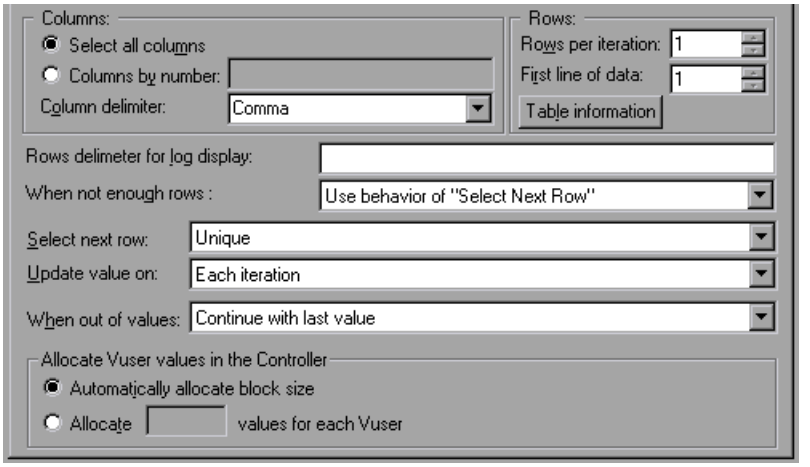
**6** If you chose **Unique** as the Data Assignment method (in step 4):

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

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

# Setting Properties for Table Type Parameters

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




**To set the Table type parameter properties:**

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

To choose all columns, select **Select all columns.**

To specify one or more columns by their number, select **Columns by number** and enter the column numbers separated by a comma or a dash. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, select 1.

**2** In the **Column delimiter** box, select a column delimiter—the character used to separate the columns in the table. The available delimiters are: comma, tab, space.

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

---

**Note:** This only relevant when the **Update value on** field is set to **Each iteration**. If **Update value on** is set to **Once**, then the same rows will be used for all iterations. See step 8.

---

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

**5** Specify a row delimiter for your data presentation in the **Rows delimiter for log display** box. This delimiter is used to differentiate between rows in the output logs. If you enable parameter substitution logging, VuGen sends the substituted values to the Replay log. The row delimiter character in the Replay log indicates a new row.

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

**7** Select a Data Assignment method from the **Select next row** list to instruct the Vuser how to select the table data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Type Parameters" on page 144.

**8** Choose an Update method from the **Update value on** list. The options are **Each Iteration** or **Once**. For more information, see "Data Assignment and Update Methods for File/Table Parameters" on page 146.

**9** If you chose to assign data using the **Unique** method (in step 7):

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

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

## Choosing Data Assignment Methods for File/Table Type Parameters

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

- Sequential
- Random
- Unique

### Sequential

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

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

### Random

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

When running a scenario, session step, or Business Process Monitor profile, you can specify a seed number for random sequencing. Each seed value represents one sequence of random values used for test execution. Whenever you use this seed value, the same sequence of values is assigned to the Vusers in the scenario or 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 *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, and *Business Availability Center* documentation.

### Unique

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

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

If there are not enough values in the data table, you can instruct VuGen how to proceed. For more details, see "Setting Properties for File Type Parameters" on page 140, or "Setting Properties for Table Type Parameters" on page 142.

## Data Assignment and Update Methods for File/Table Parameters

For File and Table type parameters, the Data Assignment method that you select, together with your choice of Update method, affect the values that the Vusers use to substitute parameters during the scenario or session step run.

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

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

### Examples

Assume that your table/file has the following values:

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

- If you chose to assign data using the **Sequential** method, then:
  - If you choose to update on **Each iteration**, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, and so on.
  - If you choose to update on **Each occurrence**, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, and so on.
  - If you choose to update **Once**, all Vusers take Kim for all iterations.

  If there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.
- If you chose to assign data using the **Random** method, then:
  - If you choose to update on **Each iteration**, the Vusers use random values from the table for each iteration.
  - If you choose to update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
  - If you choose to update **Once**, all Vusers take the first randomly assigned value for all the iterations.
- If you chose to assign data using the **Unique** method, then:
  - If you choose to update on **Each iteration**, for a test run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane, Ron, and Alice. The third Vuser, Ken, Julie, and Fred.
  - If you choose to update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.
  - If you choose to update **Once**, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, and so on.

### Vuser Behavior in the Controller (LoadRunner Only)

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

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

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

# 10

# Setting Parameter Properties

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

This chapter describes:

- About Setting Parameter Properties
- Setting Properties for Internal Data Parameter Types
- Setting Properties for User-Defined Functions
- Customizing Parameter Formats
- Selecting an Update Method

## About Setting Parameter Properties

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

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

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

For information on defining properties for Table or File type parameters, see Chapter 9, "File and Table Type Parameters."

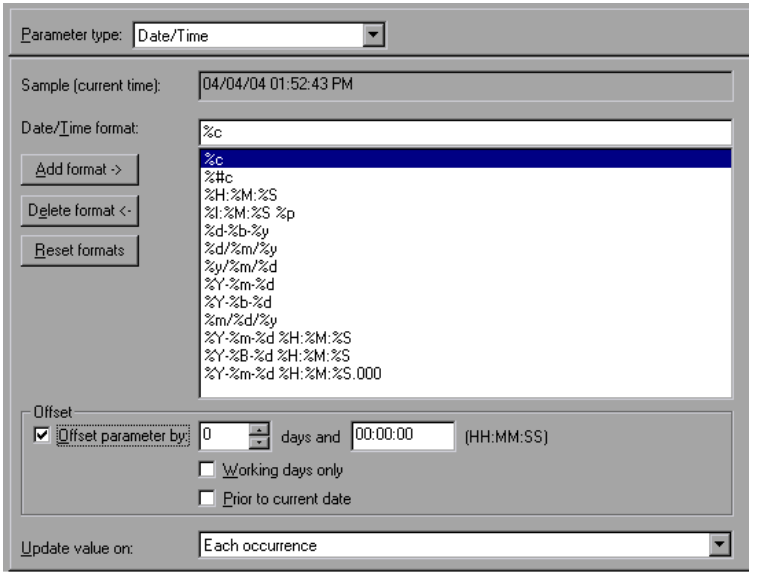
## Setting Properties for Internal Data Parameter Types

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

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

## Date/Time

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




VuGen lets you set an offset for the date/time parameter. For example, if you want to test a date next month, you set the date offset to 30 days. If you want to test your application for a future time, you specify a time offset. You can specify a forward, future offset (default) or a backward offset, a date or time that already passed. In addition, you can instruct VuGen to use date values for work days only, excluding Saturdays and Sundays.

The following table describes the date/time symbols:

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

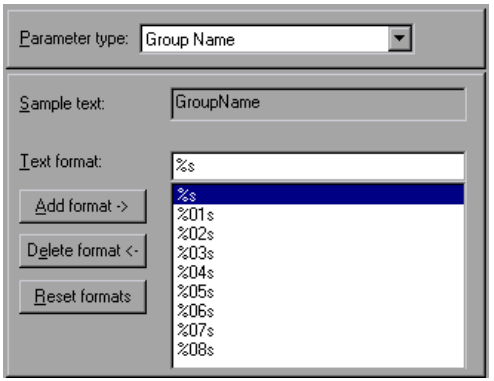
**To set the properties for Date/Time parameters:**

1. Select one of the existing date/time formats or create a new format. You can view a sample of how VuGen will display the value, in the **Sample (Current time)** box. For information on customizing parameter formats, see "Customizing Parameter Formats" on page 161.
2. To set a date and time offsets, select **Offset Parameter by** and specify the desired offset for the date and time values.

   To instruct VuGen to use working day dates only, excluding weekends, select **Working days only**. To indicate a negative offset to test a date prior to the current, select **Prior to current date**.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 162.
4. Click **Close** to accept the settings and close the Parameter Properties dialog box.

## Group Name

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

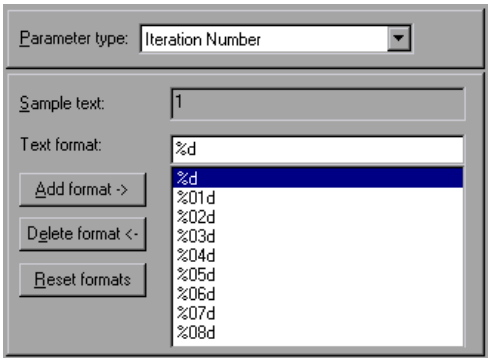



**To set properties for the Group Name parameter type:**

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 "Customizing Parameter Formats" on page 161.
2. Click **Close** to accept the settings and close the Parameter Properties dialog box.

### Iteration Number

Iteration Number replaces the parameter with the current iteration number.

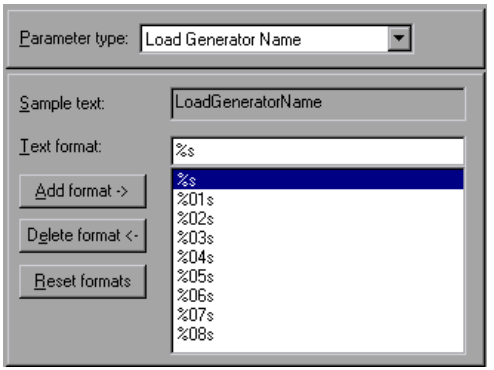



**To set the properties for the Iteration Number parameter type:**

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 161.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

## Load Generator Name

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



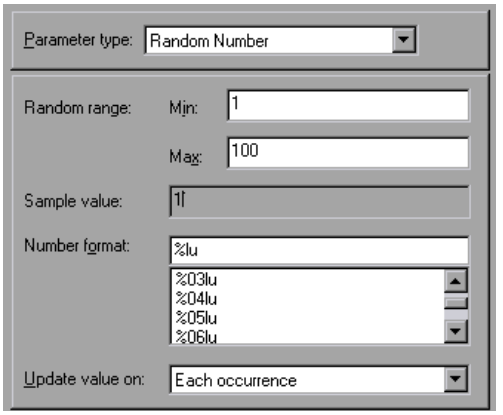

**To set the properties for the Load Generator Name parameter type:**

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 161.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

### Random Number

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

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




**To set the properties for the Random Number parameter type:**

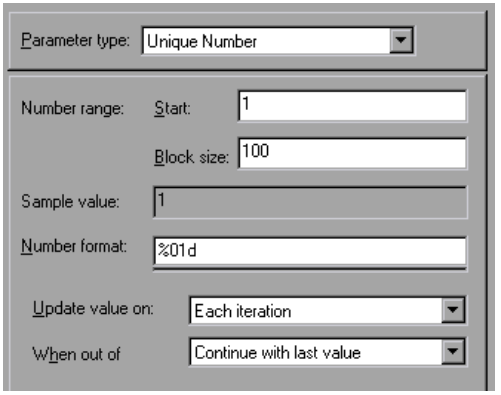
1. Enter a range defining the set of possible parameter values. You specify minimum and maximum values for the range of random numbers.
2. Select a **Number format**, indicating the length of the random number. Specify **%01lu** (or **%lu**) for one digit, **%02lu** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 162.
4. Click **Close** to accept the settings and close the Parameter Properties dialog box.

## Unique Number

Unique Number replaces the parameter with a unique number.

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

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




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

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

---

**Note:** VuGen creates only one instance of Unique Number type parameters. If you define multiple parameters and assign them the Unique Number Parameter type, the values will not overlap. For example, if you define two parameters with blocks of 100 for 5 iterations, the Vusers in the first group 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 the Unique Number parameter type:**

1. Enter a start number and the desired block size. For example, if you want 500 numbers beginning with 1, specify 1 in the **Start** box, and 500 in the **Block size per Vuser** box.
2. Select a Number format, indicating the length of the unique number. Specify **%01d** (or **%d**) for one digit, **%02d** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 162.
4. Indicate what to do when there are no more unique values, in the **When out of values** box: **Abort Vuser**, **Continue in cyclic manner**, or **Continue with last value**.

---

**Note:** (For LoadRunner users!)

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

---

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

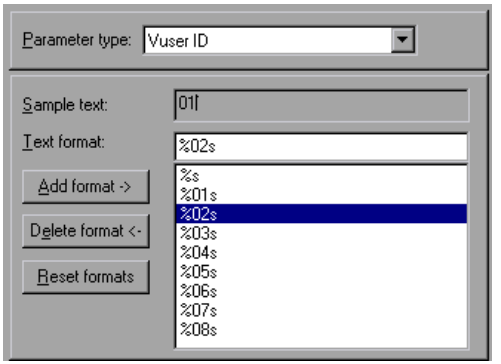
## Vuser ID

---

**Note:** This parameter type applies primarily to LoadRunner users.

---

Vuser ID replaces the parameter with the ID number assigned to the Vuser by the Controller during a scenario run, or the Console during a session step run. When you run a script from VuGen, the Vuser ID is always -1.




---

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

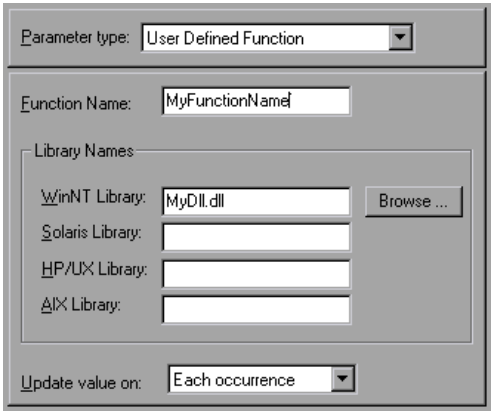
---

**To set the properties for the Vuser ID parameter type:**

1. Select one of the available formats or create a new one. You select a format to specify the length and structure of the parameter string. For details, see "Customizing Parameter Formats" on page 161.
2. Click **Close** to accept the settings and close the Parameter Properties dialog box.

# Setting Properties for User-Defined Functions

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




**To set the properties for user-defined functions:**

1. Specify the function name in the **Function Name** box. Use the name of the function as it appears in the DLL file.
2. In the **Library Names** section, specify a library in the relevant **Library** box. If necessary, locate the file using the **Browse** command.
3. Select an update method for the values. For more information on update methods for user-defined functions, see “Selecting an Update Method” on page 162.

# Customizing Parameter Formats

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

---

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

---

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

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

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

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

**To add a format to a parameter type:**

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

---

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

---

**To delete a format:**

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

**To restore the original formats:**

Click **Reset formats**.

## Selecting an Update Method

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

- Each Occurrence
- Each Iteration
- Once

### Each Occurrence

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

### Each Iteration

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

---

**Note:** If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. For more information about action blocks, see "Creating Action Blocks" on page 177.

---

### Once

The **Once** method instructs the Vuser to update the parameter value only once during the scenario or 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.

# 11

# Correlating Statements

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

This chapter describes:

- About Correlating Statements
- 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 except for GUI.***

## About Correlating Statements

The primary reasons for correlating statements are:

- **to simplify or optimize your code**

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

➤ **to generate dynamic data**

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

➤ **to accommodate unique data records**

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

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

The main steps in correlating a script are:

**1 Determine which value to correlate.**

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

Alternatively, you can use the *WDiff* utility distributed with VuGen to determine the inconsistencies within your script. For more information, see "Comparing Vuser Scripts using WDiff" on page 170.

**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 19, "Correlating Java Scripts."
- For Database Vusers, see Chapter 24, "Correlating Database Vuser Scripts."
- For Web Vusers, see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts."
- For COM Vusers, see Chapter 31, "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 169.

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

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

### Using lr_eval_string

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

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

### Using lr_save_string

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

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

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

# Using Correlation Functions for Java Vusers

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

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

When recording a CORBA-Java or RMI-Java script, VuGen performs correlation internally. For more information, see Chapter 19, "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

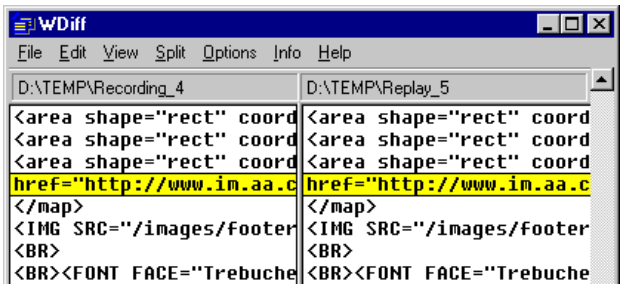
A useful tool in determining which values to correlate is *WDiff*. This utility lets you compare recorded scripts and results to determine which values need to be correlated.

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.

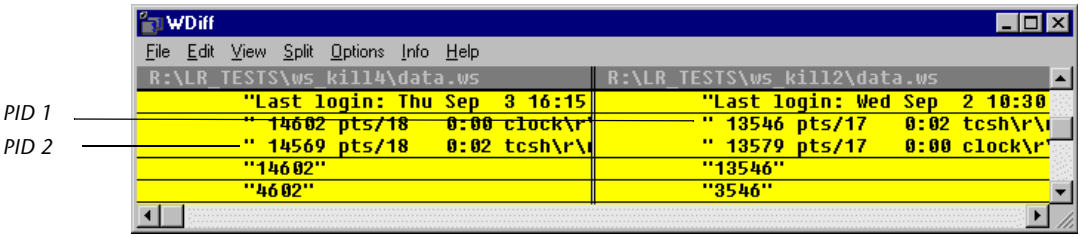
To use WDiff effectively, you record the identical operation twice, and compare the scripts (or data files for Tuxedo, WinSock, and Jolt). WDiff displays differences in yellow. Note that not all differences indicate a value to correlate. For example, certain receive buffers that indicate the time of 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*, and so forth).
4. Select **Tools** > **Compare with Vuser**. The Open Test box opens.
5. Specify a Vuser script for comparison (other than the one in the current VuGen window) and click **OK**. WDiff opens and the differences between the Vuser scripts are highlighted in yellow.



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




**7** Determine which values need to be correlated.

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

To continue with correlation, refer to the appropriate section:

- For Java language Vusers, see Chapter 19, "Correlating Java Scripts."
- For Database Vusers, see Chapter 24, "Correlating Database Vuser Scripts."
- For Web Vusers, see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts."
- For COM Vusers, see Chapter 31, "Understanding COM Vuser Scripts."
- For Tuxedo Vusers, see Chapter 69, "Developing Tuxedo Vuser Scripts."
- For WinSock Vusers, see Chapter 27, "Working with Windows Socket Data."

## Modifying Saved Parameters

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

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

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

# 12

# 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:

- About Run-Time Settings
- Configuring Run Logic Run-Time Settings (multi-action)
- Pacing Run-Time Settings
- 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 except for GUI.***

# About Run-Time Settings

After you record a Vuser script, you can configure its run-time settings. The run-time settings define the way that the script runs. These settings are stored in the file *default.cfg,* located in the Vuser script directory. Run-time settings are applied to Vusers when you run a script using VuGen, the Controller, Tuning Console, or Administration 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 LoadRunner Controller or the Tuning Module. For more information, refer to the product's documentation.

---

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

**Think Time:** Off in VuGen and Replay as Recorded in the Controller.

**Log:** Standard in VuGen and off in the Controller.

**Download non-HTML resources:** Enabled in VuGen and the Controller.

---

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

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

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

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

---

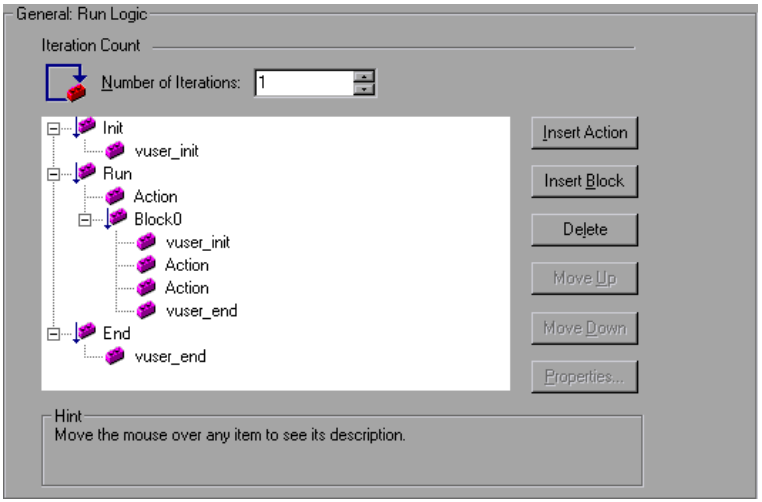
**Note:** The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 183.

---

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

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

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




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

---

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

---

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

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

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

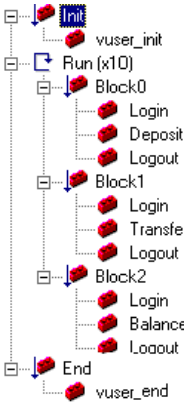
**Iterations:** In addition to setting the number of iterations for the entire *Run* section, you can set iterations for individual actions or action blocks. This is useful, for example, in emulating a commercial site where you perform many queries to locate a product, but only one purchase.

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

## Creating Action Blocks

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

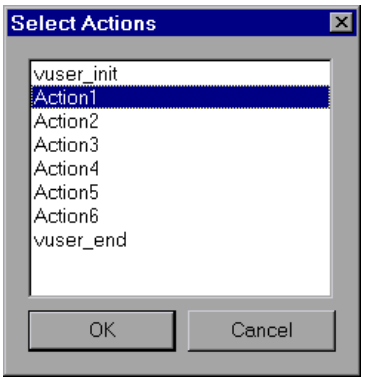
In the following example, *Block0* performs a deposit, *Block1* performs a transfer, and *Block2* submits a balance request. The *Login* and *Logout* actions are common to the three blocks.



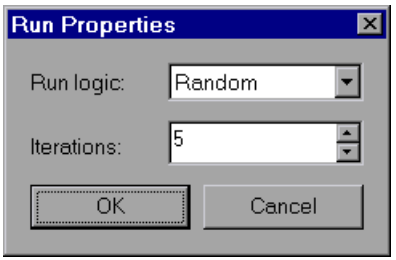

You configure each block independently—its sequence and iterations.

**To configure actions and action blocks:**

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




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

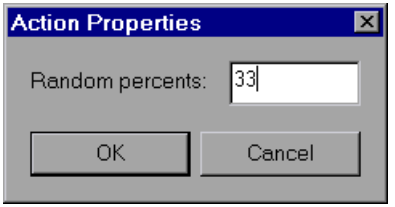


**10** Select *Sequential* or *Random* from the **Run Logic** list, indicating to VuGen whether to run the actions sequentially or randomly.

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

**12** Click **OK**.

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




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

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

# Pacing Run-Time Settings

---

**Note:** The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 183.

---

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

- As soon as the previous iteration ends.
- 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.

When you run the script, VuGen shows the time the Vuser waited between the end of one iteration and the start of the next one, in the Execution Log.

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

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

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

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

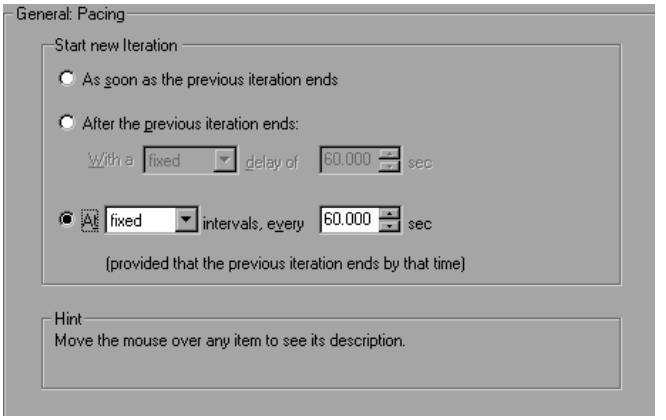
For further instruction about setting the Pacing options, see "Configuring Pacing Run-Time Settings (multi-action)" on page 182.

# Configuring Pacing Run-Time Settings (multi-action)

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

**To set the pacing between iterations:**

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




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

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

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

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

**4** For the **At … intervals** option:

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

**5** Click **OK**.

# Setting Pacing and Run Logic Options (single action)

---

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

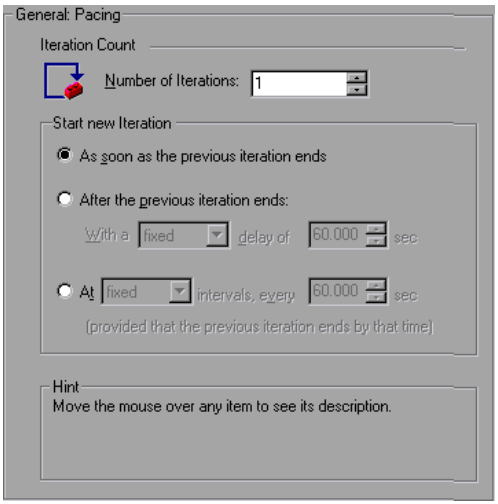
---

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

**To set the iteration and pacing preferences:**



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




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

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

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

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

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

5 For the **At … intervals** option:

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

6 Click **OK**.

For an overview of the pacing options, see "Pacing Run-Time Settings" on page 180.

## Configuring the Log Run-Time Settings

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

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

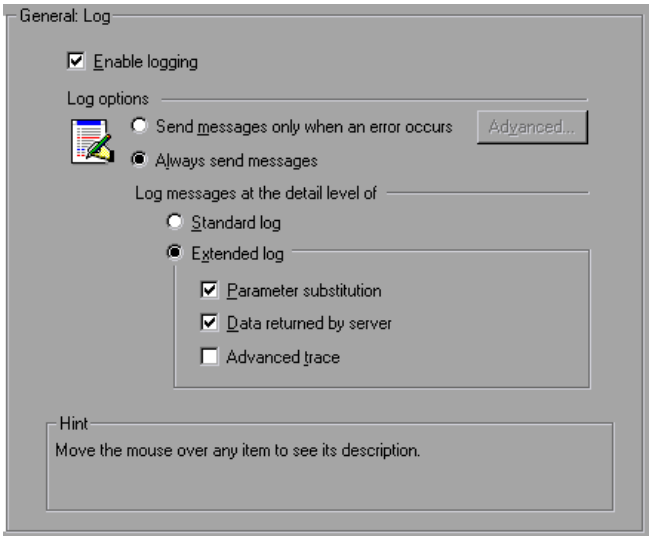
---

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

---



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




### Enable Logging

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

### Log Options

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

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

If you choose to send messages only when errors occur, also known as JIT, (Just in Time) messaging, you can set an advanced option, indicating the size of the log cache. See "Setting the Log Cache Size" on page 187.

### Setting the Log Detail Level

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

---

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

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

---

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

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

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

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

- **Parameter substitution:** Select this option to log all parameters assigned to the script along with their values. For more information on parameters, see Chapter 8, "Working with VuGen Parameters."
- **Data returned by server:** Select this option to log all of the data returned by the server.
- **Advanced trace:** Select this option to log all of the functions and messages sent by the Vuser during the session. This option is useful when you debug a Vuser script.

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

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

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

## Setting the Log Cache Size

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

**The following is the sequence of the logging:**

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

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

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

### Logging CtLib Server Messages

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

```
LRD_CTLIB_DB_SERVER_MSG_LOG;
```

VuGen logs all server messages in the Standard log.

To send the server messages to the output (in addition to the Standard log), type:

```
LRD_CTLIB_DB_SERVER_MSG_ERR;
```

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

```
LRD_CTLIB_DB_SERVER_MSG_NONE;
```

---

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

---

## Configuring the Think Time Settings

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

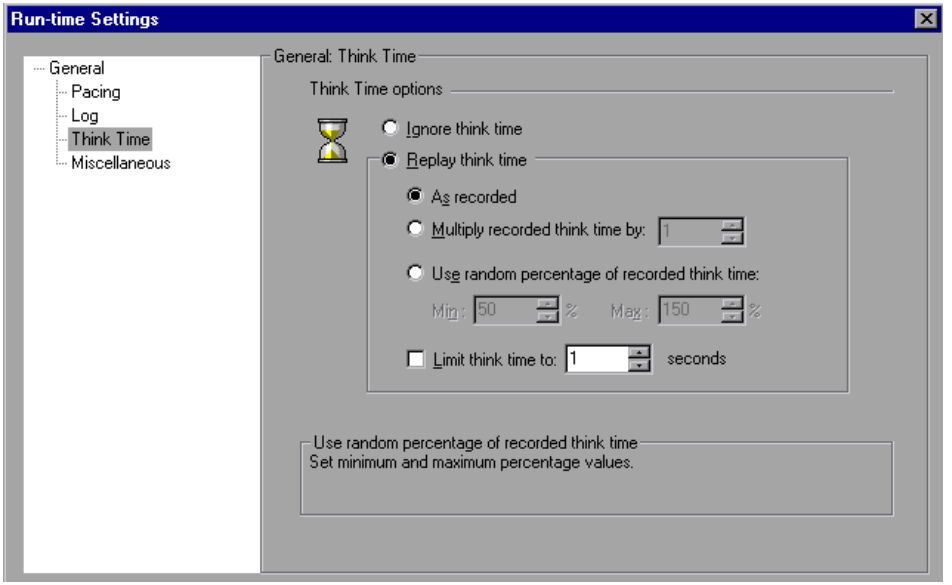
```
lr_think_time(8);
```

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

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



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




## Think Time Options

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

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

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

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

- **Multiply recorded think time by:** During replay, use a multiple of the recorded think time. This can increase or decrease the think time applied during playback. For example, if a think time of four seconds was recorded, you can instruct your Vuser to multiply that value by two, for a total of eight seconds. To reduce the think time to two seconds, multiply the recorded time by 0.5.
- **Use random percentage of the recorded think time:** Use a random percentage of the recorded think time. You set a range for the think time value by specifying a range for the think time. For example, if the think time argument is 4, and you specify a minimum of 50% and a maximum of 150%, the lowest think time can be two (50%) and the highest value six (150%).
- **Limit think time to:** Limit the think time's maximum value.

## Configuring Additional Attributes Run-Time Settings

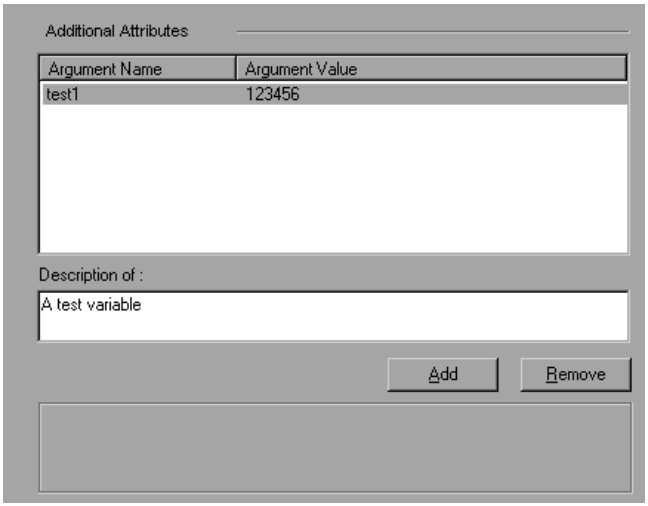
For the Mercury Tuning Module, you can use the Additional Attributes node to provide additional arguments for a Vuser script. The Additional Attributes settings apply to all Vuser script types.

You specify command line arguments that you can retrieve at a later point during the test run, using **lr_get_attrib_string**. Using this node, you can pass parameters to prepared scripts, enabling you to test and monitor your servers with different client parameters.

**To set additional attributes:**



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




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

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

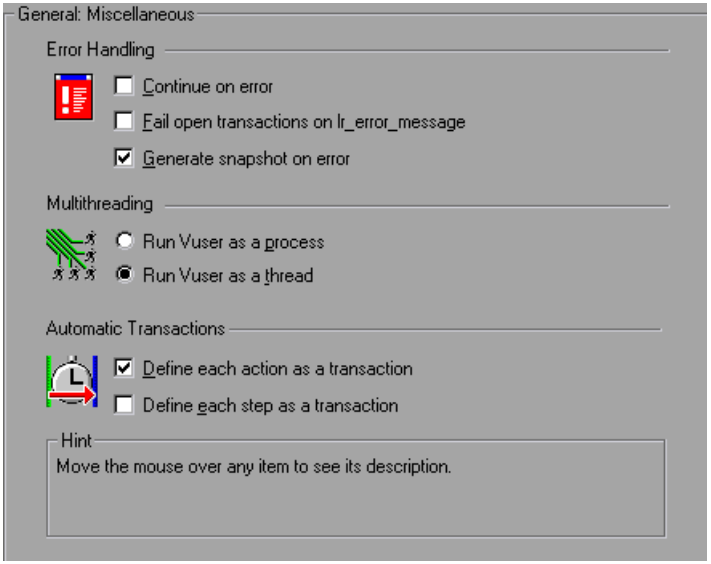
## Configuring Miscellaneous Run-Time Settings

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

- Error Handling
- Multithreading
- Automatic Transactions



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




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

## Error Handling

**Continue on Error:** This setting instructs Vusers to continue script execution when an error occurs. This option is turned off by default, indicating that the Vuser will exit if an error occurs.

**Fail open transactions on lr_error_message:** This option instructs VuGen to mark all transactions in which an **lr_error_message** function was issued, as *Failed*. The **lr_error_message** function is issued through a programmed *If* statement, when a certain condition is met.

**Generate Snapshot on Error:** This option generates a snapshot when an error occurs. You can see the snapshot by viewing the Vuser Log and double-clicking on the line at which the error occurred.

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

### Error Handling for Database Vusers

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

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

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

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

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

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

VuGen supports the following severity levels:

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

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

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

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

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

---

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

---

### Error Handling for RTE Vusers

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

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

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

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

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

## Multithreading

Vusers support multithread environments. The primary advantage of a multithread environment is the ability to run more Vusers per load generator. Only threadsafe protocols should be run as threads. (not applicable to Business Availability Center)

---

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

---

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

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

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

### Automatic Transactions

You can instruct LoadRunner or the Tuning Console (not applicable to Business Availability Center) to handle every step or action in a Vuser script as a transaction. This is called using automatic transactions. LoadRunner or the Tuning Console assigns the step or action name as the name of the transaction. By default, automatic transactions per action are enabled.

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

If you disable automatic transactions, you can still insert transactions manually during and after recording. For more information on manually inserting transactions, see Chapter 7, “Enhancing Vuser Scripts.”

---

**Note:** If you require the Vusers to generate breakdown data for diagnostics (J2EE) during the scenario run, do not use automatic transactions. Instead, manually define the beginning and end of each transaction.

---

# Setting the VB Run-Time Settings

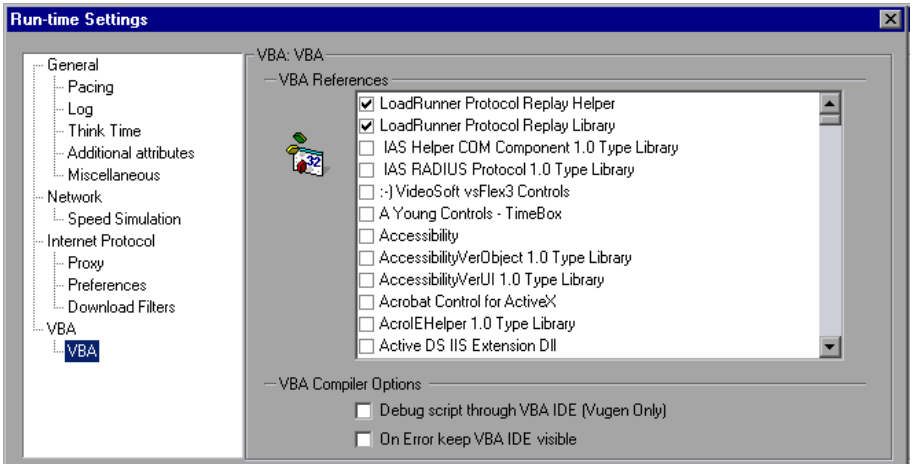
Before running your Visual Basic script, you indicate which libraries to reference during replay. VuGen displays a list of all of the libraries stored on the machine.

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



**To set the VBA Run-Time settings:**

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




**2** In the **VBA References** section, select the reference library that you want to use while running the script. Select a library to display its description and version in the bottom of the dialog box.

**3** Select the appropriate compiler options:

Select **Debug script through VBA IDE** to enable debugging through the Visual Basic IDE (Integrated Development Environment).

Select **On Error keep VBA IDE visible** to keep the Visual Basic IDE visible during script execution.

**4** Choose **OK** to apply the run-time settings.

# 13

# Configuring Network Run-Time Settings

To simulate the speed over a network, you configure the Network run-time settings.

This chapter describes:

- About Network Run-Time Settings
- Setting the Network Speed

***The following information applies to all Internet Protocol Vuser types, Citrix ICA, Oracle NCA, and WinSock.***

For information about the general run-time settings that apply to all Vusers, see Chapter 12, "Configuring Run-Time Settings."

## About Network 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 Interest 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:**



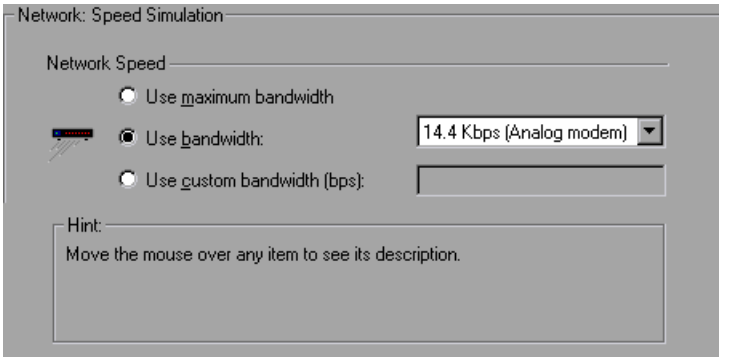
- Click the **Run-Time Settings** button on the VuGen toolbar.
- Press the keyboard shortcut key **F4**.

➤ Choose **Vuser** > **Run-Time Settings**.

Note that you can also modify the run-time settings from the LoadRunner Controller or the Mercury Tuning Module. For more information, refer to your product's documentation.

## Setting the Network Speed

You use the **Network:Speed Simulation** node in the Run-Time Settings tree, to set the modem emulation for your tuning or testing environment.




### Speed Simulation

Using the Speed Simulation settings, you can select a bandwidth that best emulates the environment under test. The following options are available:

**Use maximum bandwidth**: By default, bandwidth emulation is disabled and the Vusers run at the maximum bandwidth that is available over the network.

**Use bandwidth:** Indicate a specific bandwidth level for your Vuser to emulate. You can select a speed ranging from 14.4 to 512 Kbps, emulating analog modems, ISDN, or DSL.

**Use custom bandwidth:** Indicate a bandwidth limit for your Vuser to emulate. Specify the bandwidth in bits, where 1 Kilobit=1024 bits.

# 14

# Running Vuser Scripts in Standalone Mode

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

This chapter describes:

- About Running Vuser Scripts in Standalone Mode
- Running a Vuser Script in VuGen
- Viewing the Replay Log
- Using VuGen's Debugging Features
- Using VuGen's Debugging Features for Web Vuser Scripts
- Working with VuGen Windows
- Running a Vuser Script from a Command Prompt
- Running a Vuser Script from a UNIX Command Line
- Integrating Scripts into Tests

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

## About Running Vuser Scripts in Standalone Mode

After creating a script, you check its functionality by running it in standalone mode, directly from the VuGen interface. If the script is UNIX-based, you run it from a UNIX command line. To run GUI Vusers in standalone mode, use WinRunner.

When the standalone execution is successful, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

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

- enhance the script with Vuser functions (see Chapter 7, "Enhancing Vuser Scripts")
- parameterize the script (see Chapter 8, "Working with VuGen Parameters")
- correlate queries in the script (see Chapter 11, "Correlating Statements")

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

## Running a Vuser Script in VuGen

After developing a Vuser script, run it using VuGen to verify that it executes correctly. You can set several options for replay.

---

**Note:** VuGen runs Vuser scripts on Windows platforms only. To run UNIX-based Vuser scripts, see "Running a Vuser Script from a UNIX Command Line" on page 218.

---

## Configuring Replay Options

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

**Animated run delay:** The time delay in milliseconds between commands. The default delay value is 0.

**Only animate functions in Actions sections:** Only animates the content of the Action sections, but not the *init* or *end* sections.

**Prompt for results directory:** Prompts you for a results directory before running a script from VuGen. If this option is not selected, VuGen automatically names the directory *result1*. Subsequent script executions will automatically overwrite previous ones unless you specify a different result file. Note that results are stored in a subdirectory of the script.

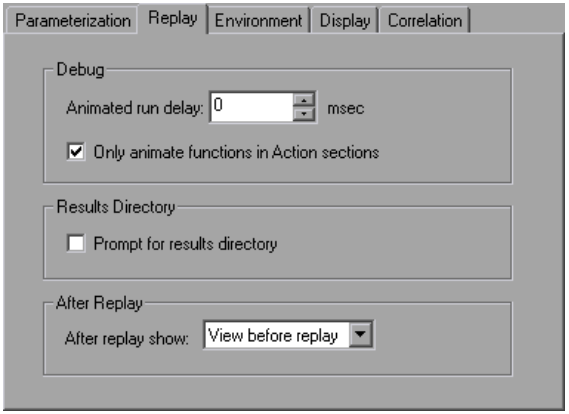
**After replay show:** Instructs VuGen how to proceed after the replay:

- **View before replay:** Return to the view you had before replay.
- **Replay summary:** Go directly to the Replay Summary window in the Workflow Wizard.
- **Visual Test Results**: Open the Test Results Summary. (This is the same as choosing **View** > **Test Results** after replay.)

**To enable animation and set its properties:**

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

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

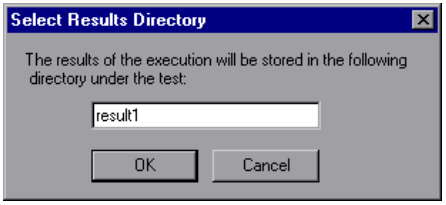



3 Select the **Replay** tab.

4 In the **Animated run delay** box, specify a delay in milliseconds and click **OK**.

5 Select **Only animate functions in Actions sections** to animate only the content of the Action sections.

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




7 Type a directory name for the execution results, or accept the default name and click **OK**.

### Setting the Display Options

If you are running a Web Vuser script, you can set the Display options (**Tools** > **General Options**). These options specify whether to display VuGen's run-time viewer, whether to generate a report during script execution, and so forth.

For more information, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 215.

## Replaying a Vuser Script

Before you integrate a script into a test or production environment, you run it from VuGen to make sure it is functional. VuGen provides several tools that allow you to monitor the replay and locate any existing and potential problems. These include:

- Viewing the Replay Log
- The Run-Time Data Tab
- The Run Step by Step Command
- Breakpoints
- Bookmarks
- Go To Commands

**To replay a script in VuGen:**

1. Select **Vuser** > **Run**.

   The Output window opens at the bottom of the VuGen main window—or clears, if already open—and VuGen begins executing the Vuser script. In tree view, VuGen runs the Vuser script from the first icon in the script. In Script view, it runs the Vuser script from the first line of the script.

2. Click the Output window's **Replay Log** tab for a log of all of the actions of the Vuser, along with warnings and errors. For more information, see "Viewing the Replay Log" on page 208.

3 To view a summary of the run-time data and the parameters as they are being used, see the Output window's **RunTime Data** tab. For more information, see "The Run-Time Data Tab" on page 209.

4 To hide the Output window during or after a script run, select **View** > **Output Window**. VuGen closes the Output window and removes the check mark from next to **Output Window** on the **View** menu.

5 To interrupt a Vuser script that is running, select **Vuser** > **Pause**, to temporarily pause the script run, or **Vuser** > **Stop**, to end the script run.

### Viewing the Replay Log

The Output window's Replay Log displays messages that describe the actions of the Vuser as it runs. This information tells you how the script will run when executed in a scenario, session step, or profile.

When script execution is complete, you examine the messages in the Replay Log to see whether your script ran without errors.

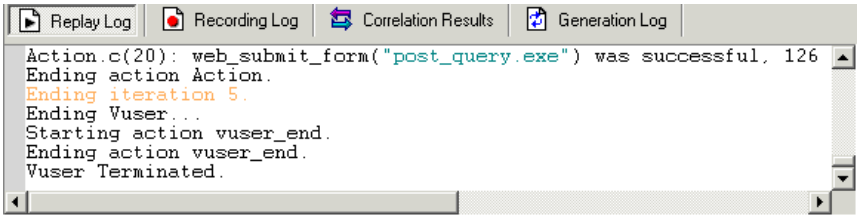
Various colors of text are used in the Replay Log.

- **Black**: Standard output messages
- **Red**: Standard error messages
- **Green**: Literal strings that appear between quotation marks (e.g. URLs)
- **Blue**: Transaction Information (starting, ending, status and duration)
- **Orange:** The beginning and ending of iterations.

If you double-click on a line beginning with the Action name, the cursor jumps to the step within the script that generated.

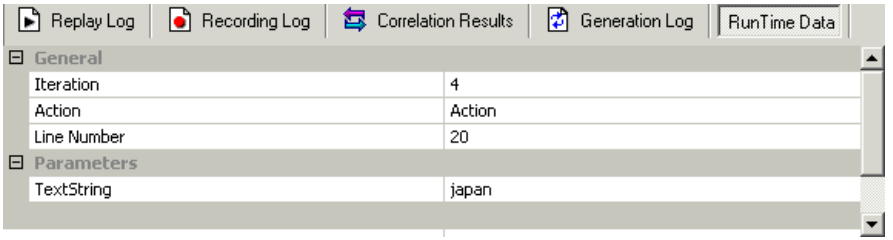
For more information on closing and opening the Output window, see "Replaying a Vuser Script" on page 207.

The following example shows Replay Log messages from a Web Vuser script run.




## The Run-Time Data Tab

You can track the script information that becomes updates during replay, using the Run Time Data tab.




During replay, click the rightmost tab, **RunTime Data**. The tab contains two expandable/collapsible sections:

**General:** The general section shows the current iteration number, the Action name of the currently replayed step, and the line number within the script (Script view).

**Parameters:** The parameters section shows all parameters defined with the script and their substitution values based on the selected update method (sequential, unique, etc.). VuGen shows this information even if the parameter is not used in the script. For more information, see Chapter 8, “Working with VuGen Parameters.”

Note that the RunTime Data tab is not accessible after the test run, since it only displays data that changes during replay.

# Using VuGen's Debugging Features

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

VuGen contains additional features to help debug Web Vuser scripts. For details, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 215.

**To view the Debug toolbar:**

Right-click the toolbar area and select **Debug**. The Debug toolbar appears in the toolbar area.



## The Run Step by Step Command

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

**To run the script step by step:**



1 Select **Vuser** > **Run Step by Step**, or click the **Step** button on the Debug toolbar.

VuGen executes the first line of the script.

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

## Breakpoints

Breakpoints pause execution at specific points in the script. This enables you to examine the effects of the script on your application at pre-determined points during execution. To manage the bookmarks, use the "The Breakpoint Manager" on page 212.

**To set breakpoints:**

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



2 Select **Insert** > **Toggle Breakpoint**, or click the **Breakpoint** button in the Debug toolbar. Alternatively, press F9 on the keyboard. The Breakpoint symbol () appears in the left margin of the script.





3 To disable a breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Enable / Disable Breakpoint** button on the Debug toolbar. A white dot inside the Breakpoint symbol indicates a disabled breakpoint. When one breakpoint is disabled, script execution is paused at the following breakpoint. Click the button again to enable the breakpoint.

To remove the breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Breakpoint** button or press F9.

**To run the script with breakpoints:**

1 Begin running the script as you normally would.

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

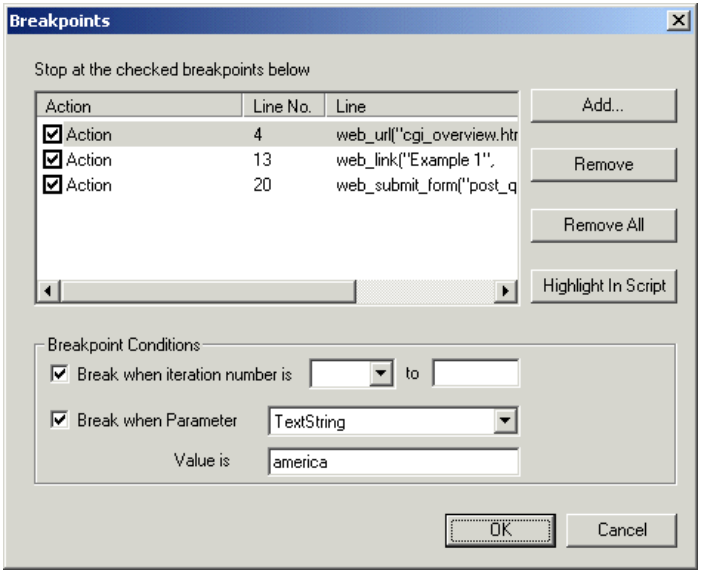
2 To resume execution, select **Vuser** > **Run**.

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

## The Breakpoint Manager

You can view and manage breakpoints using the Breakpoint Manager. From the Breakpoint Manager you can manipulate al of the breakpoints in your script.

To open the Breakpoint Manager, select **Edit** > **Breakpoints**.




**To jump to the breakpoint location in the script:**

1 Select a breakpoint from the list.

2 Click **Highlight in Script**. The line in the script becomes highlighted.

Note that you can only highlight one breakpoint at a time.

## Managing Breakpoints

From the Breakpoint Manager, you can add, remove, disable, or conditionalize a breakpoint.

**To add a breakpoint:**

1 Click **Add**. The Add Breakpoint dialog box opens.

2 Select an **Action** and specify the **Line** number where you want add the breakpoint.

3 Click **OK.** The Breakpoint is added to the list of breakpoints.

**To remove a breakpoint:**

1 To remove a single breakpoint, select the breakpoint and click **Remove**.

2 To remove all the breakpoints at once, click **Remove All**.

**To enable/disable a breakpoint:**

1 To enable a breakpoint, in the Action column, select the action's check box.

2 To disable a breakpoint, in the Action column, clear the action's check box.

From the Breakpoint Manager, you can set breakpoints to pause execution under certain conditions.

**To conditionalize a breakpoint:**

1 To pause the script after a specific number of iterations, select **Break when iteration number is** and enter the desired number.

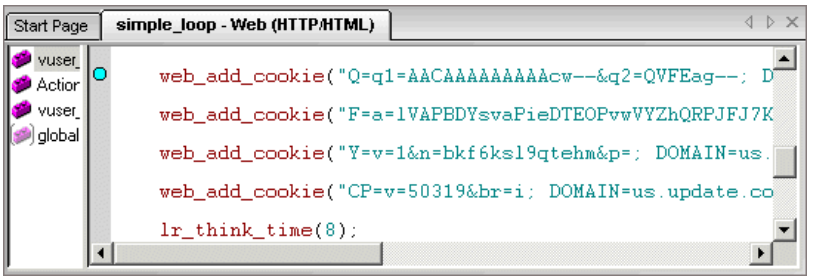
2 To pause the script when parameter *X* has a specific value, select **Break when Parameter** *X* **Value is** and enter the desired value. For more information about parameters, see Chapter 8, "Working with VuGen Parameters."

## Bookmarks

When working in Script view, VuGen lets you place bookmarks at various locations within your script. You can navigate between the bookmarks to analyze and debug your code.

**To create a bookmark:**

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




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

3 To move between bookmarks:

To move to the next bookmark, click F2.

To navigate to the previous bookmark, click Shift + F2.

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

---

**Note:** You can only navigate between bookmarks in the current action. To navigate to a bookmark in another action, select that action in the left pane and then press F2.

---

### Go To Commands

To navigate around the script without using bookmarks, you can use the Go To command. Choose **Edit > Go To Line** and specify the line number of the script. This navigation is also supported in Tree view.

If you want to examine the Replay log messages for a specific step or function, select the step in VuGen and choose **Edit > Go To Step in Replay Log.** VuGen places the cursor at the corresponding step in the Output window's Replay Log tab.

## Using VuGen's Debugging Features for Web Vuser Scripts

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

- You can instruct VuGen to display a run-time viewer when you run a Web Vuser script. The run-time viewer 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.Chapter 50, "Power User Tips for Web Vusers."

- You can specify whether or not a Web Vuser generates a Results Summary report during script execution. The Results Summary report summarizes the success or failure of each step in the Web Vuser scripts and allows you to view the Web page returned by each step. For additional details on working with the Results Summary report, choose **View > Test Results** and click F1 to open the online help.Chapter 49, "Using Reports to Debug Vuser Scripts."

---

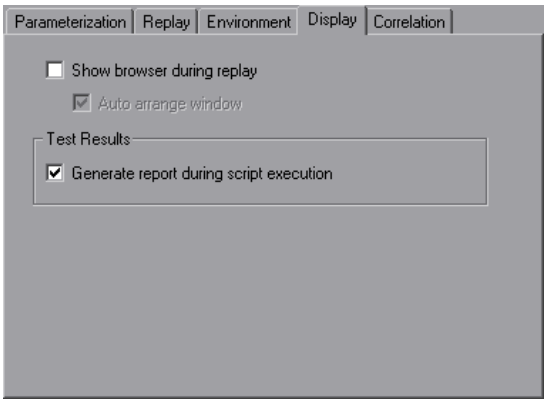
**Note:** Transaction times may be increased when a Vuser generates a Results Summary report.

Vusers can generate Results Summary reports only when run from VuGen. When you run a script from the Controller, Tuning Module, or Administration Console, Vusers do not 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 browser during replay** check box to 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. You can open the report after script execution by selecting **View** > **Test Results**.

4 Click **OK** to accept the settings and close the General Options dialog box.

## Working with VuGen Windows

You can show and rearrange VuGen's windows to view the relevant data for your script, using the following features:

➤ **Show/Hide the Output Window**

Select **View** > **Output Window** to show and hide the Output window below the VuGen script editor. The Output window has several tabs, depending on the protocol. The most common tabs are the Replay Log, Recording Log, Generation Log, and Correlation Results. For more information, see "Viewing the Replay Log" on page 208.

➤ **Display All Thumbnails**

Select **View** > **Show All Thumbnails** to show all of the script's steps as thumbnails. To show thumbnails for primary steps only, clear this option. For more information, see "Viewing Script Thumbnails" on page 24.

➤ **Display Grids**

Select **View** > **Enable Data Grids** to enable grid display of the data in the protocols that support data grids (Database, COM, and Microsoft . NET). The grids appear inside the script.

➤ **Window Manipulation**

Select **Window** > **Close All** to close all of the open scripts. If necessary, VuGen will prompt you to save the unsaved scripts.

## Running a Vuser Script from a Command Prompt

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

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

1 Select **Start** > **Programs** > **Command Prompt** to open a **Command Prompt** window, or select **Start** > **Run** to open the Run dialog box.

2 Type the following and press **Enter**:

<*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 verify that your script runs correctly, follow these steps:

1 **Test the recorded script from VuGen.**

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

2 **Copy the Vuser script files to a UNIX drive.**

Transfer the files to a local UNIX drive.

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

For details, see "The verify_generator Test" on page 218.

4 **Test the script from the UNIX command line.**

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

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

### The verify_generator Test

The verify utility checks the local host for its communication parameters and its compatibility with all types of Vusers.

The utility checks the following items in the Vuser environment:

- at least 128 file descriptors
- proper .rhost permissions: -rw-r--r--

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

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

```
verify_generator
```

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

For detailed information about the verify checks type:

```
verify_generator [-v]
```

### Command Line Options: run_db_vuser Shell Script

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

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

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

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

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

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

**-ci** *ci_file_name*
Execute a specific .ci file.

**-out** *output_path*
Place the results in a specific directory.

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

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

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

# Integrating Scripts into Tests

Once you have successfully run a script in standalone mode to verify that it is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile.

When you integrate a test, you provide information indicating:

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

For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center,* or *Business Availability Center* documentation.

## Using VuGen to Create a LoadRunner Scenario

---

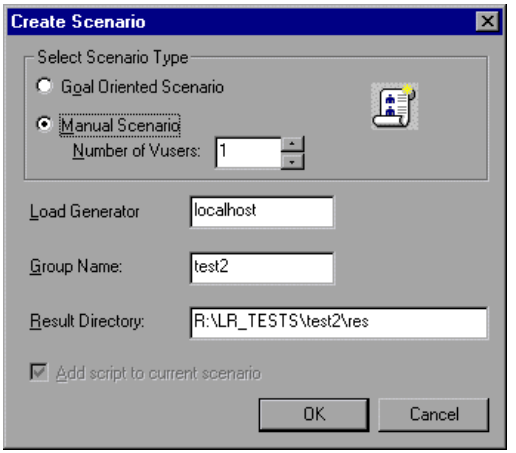
**Note:** The following section only applies to LoadRunner. For information on integrating scripts into Business Process profiles, refer to the *Business Availability Center* documentation.

---

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

**To create a scenario from VuGen:**

1 Choose **Tools** > **Create Controller Scenario**. The Create Scenario dialog box opens.




2 Choose either a goal oriented or a manual scenario.

In a goal-oriented scenario, LoadRunner automatically builds a scenario based on the goals you specify, whereas in a manual scenario, you specify the number of Vusers to run.

3 For a manual scenario, enter the number of Vusers you want to execute the script.

4 Enter the name of the machine upon which you want the Vusers to run, in the **Load Generator** box.

5 For a manual scenario, users with common traits are organized into groups. Specify a new group name for the Vusers in the **Group Name** box.

6 For a goal-oriented scenario, specify a **Script Name**.

7 Enter the desired location for the results in the **Result Directory** box.

8 If a scenario is currently open in the Controller and you want to add the script to this scenario, select the **Add script to current scenario** check box. If you clear the check box, LoadRunner opens a new scenario with the specified number of Vusers.

**9** Click **OK**. VuGen opens the Controller in the Vuser view.

**10** If you configured the Controller to save the script on a shared network drive, you may need to perform path translation.

For more information, refer to the *LoadRunner Controller User's Guide*.

# 15

# Managing Scripts Using Quality Center

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

This chapter describes:

- About Managing Scripts Using Quality Center
- Connecting to and Disconnecting from Quality Center
- Opening Scripts from a Quality Center Project
- Saving Scripts to a Quality Center Project
- Managing Script Versions in VuGen

## About Managing Scripts Using Quality Center

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

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

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

# Connecting to and Disconnecting from Quality Center

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

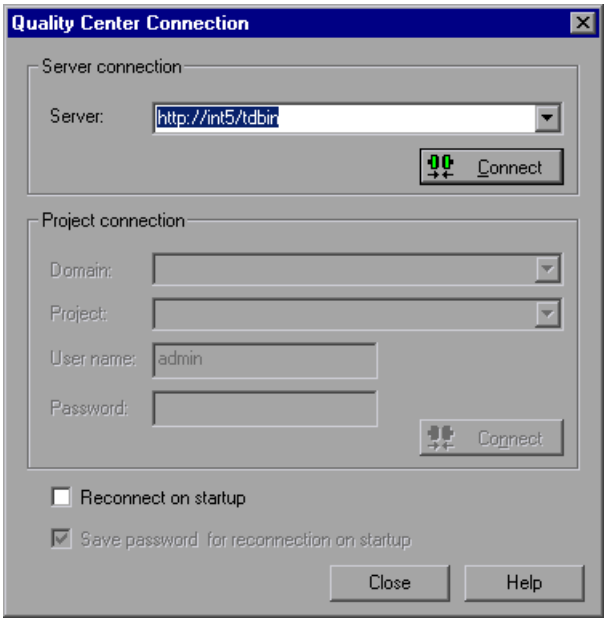
## Connecting VuGen to Quality Center

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

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

**To connect VuGen to Quality Center:**

**1** In VuGen, choose **Tools** > **Quality Center Connection**. The Quality Center Connection dialog box opens.



2 In the Server box, type the URL address of the Web server on which Quality Center 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 Domain box in the Project connection section, select a domain.

5 From the Project box in the Project connection section, select a Quality Center project.

6 In the **User Name** box, type a user name.

7 In the **Password** box, type a password.

8 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.

9 To automatically reconnect to the Quality Center server and the selected project on startup, select the **Reconnect on startup** check box.

10 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 Quality Center on startup.

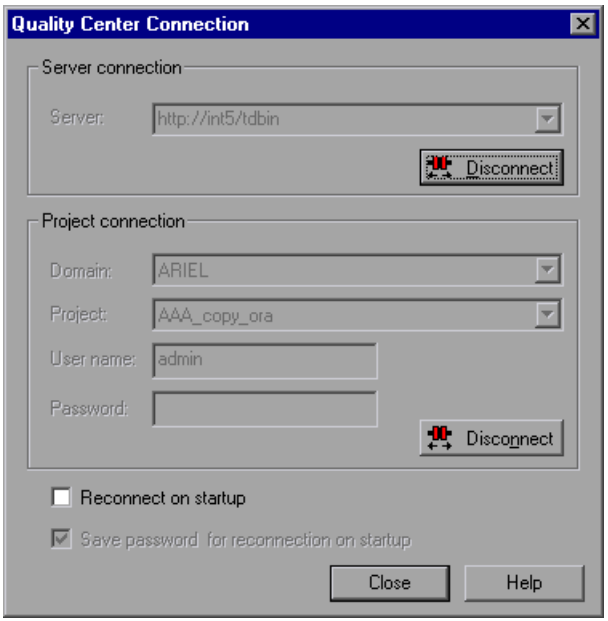
11 Click **Close** to close the Quality Center Connection dialog box.

## Disconnecting VuGen from Quality Center

You can disconnect VuGen from a selected Quality Center project and Web server.

**To disconnect VuGen from Quality Center:**

1 In VuGen choose **Tools** > **Quality Center Connection**. The Quality Center 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 Quality Center Connection dialog box.

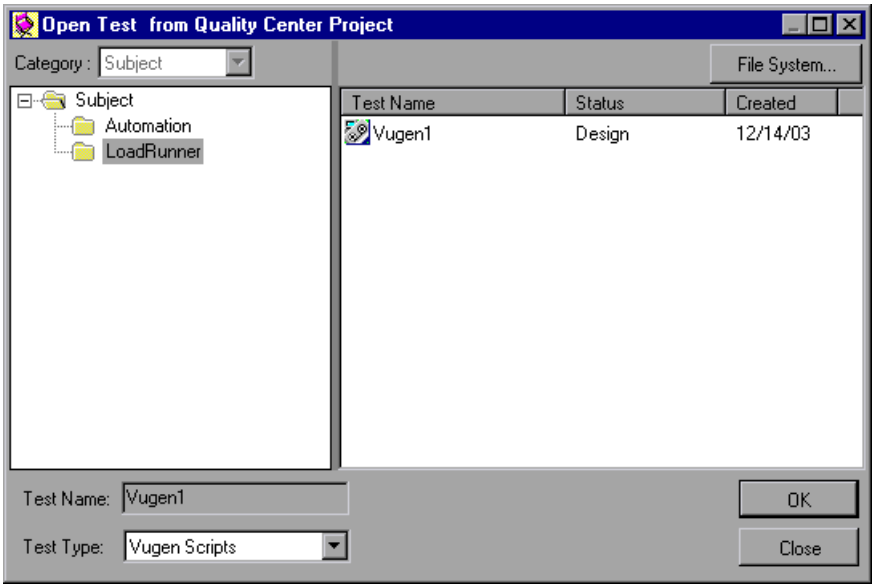
# Opening Scripts from a Quality Center Project

When VuGen is connected to a Quality Center project, you can open your scripts from Quality Center. You locate tests according to their position in the test plan tree, rather than by their actual location in the file system.

**To open a script from a Quality Center project:**

1 Connect to the Quality Center server (see "Connecting VuGen to Quality Center" on page 226).

2 In VuGen, choose **File** > **Open** or click the **File Open** button. The Open Test from Quality Center 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 Quality Center Project dialog box by clicking the **Quality Center** button.)

3 Click the relevant subject in the test plan tree. To expand the tree and view sublevels, double-click closed folders. To collapse the tree, double-click open folders.

Note that when you select a subject, the scripts that belong to the subject appear in the Test Name list.

4 Select a script from the Test Name list. The script appears in the read-only Test Name box.

5 Click **OK** to open the script. VuGen loads the script. The name of the script appears in VuGen's title bar. The **Design** tab shows all of the scripts in the test plan tree.

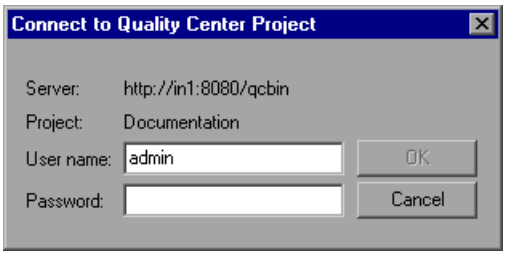
---

**Note:** You can also open scripts from the recent file list in the File menu. If you select a script located in a Quality Center project, but VuGen is currently not connected to that project, the Quality Center Connection dialog box opens. Enter your user name and password to log in to the project, and click **OK**.

---

### Opening Tests from the Recent Files List

You can open Vuser scripts from the recent files list in the File menu. If you select a script located in a Quality Center project, but VuGen is currently not connected to Quality Center or to the correct project for the script, the Connect to Quality Center Project dialog box opens and displays the correct server, project, and the name of the user who most recently opened the script on this computer.




Log in to the project, and click **OK**.

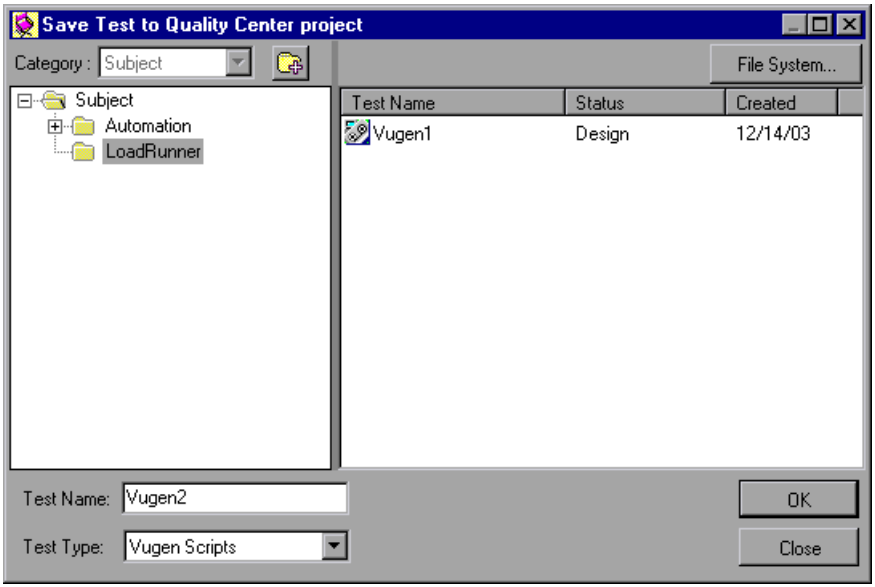
The Connect to Quality Center Project dialog box also opens if you choose to open a test that was last edited on your computer using a different Quality Center user name. You can either log in using the displayed name or you can click **Cancel** to stay logged in with your current user name.

## Saving Scripts to a Quality Center Project

When VuGen is connected to a Quality Center project, you can create new scripts in VuGen and save them directly to your project. To save a script, you give it a descriptive name and associate it with the relevant subject in the test plan tree. This helps you to keep track of the scripts created for each subject and lets you view the progress of test planning and creation.

**To save a script to a Quality Center project:**

1. Connect to the Quality Center server (see "Connecting VuGen to Quality Center" on page 226).
2. In VuGen, choose **File** > **Save As**. The Save Test to Quality Center Project dialog box opens and displays the test plan tree.



To save a script directly in the file system, click the **File System** button. The Save Test dialog box opens. (From the Save Test dialog box, you may return to the Save Test to Quality Center Project dialog box by clicking the **Quality Center** button.)

3 Select the relevant subject in the test plan tree. To expand the tree and view a sublevel, double-click a closed folder. To collapse a sublevel, double-click an open folder.

4 In the Test Name box, enter a name for the script. Use a descriptive name that will allow you identify the script easily.

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

The next time you start Quality Center, the new script will appear in Quality Center's test plan tree.

## Managing Script Versions in VuGen

When VuGen is connected to a Quality Center project with version control support, you can update and revise your automated scripts while maintaining old versions. This helps you keep track of the changes made to each script, see what was modified from one version of a script to another, or return to a previous version of the script.

You add a script to the version control data base by saving it in a project with version control support. You manage versions by checking scripts in and out of the version control database.

The script with the latest version is the script that is located in the Quality Center repository and is used by Quality Center for all test runs.

---

**Note:** The **Quality Center Version Control** options in the **File** menu are available only when you are connected to a Quality Center project database with version control support.

---

## Adding Scripts to the Version Control Database

When you use **Save As** to save a new script in a Quality Center project with version control support, QuickTest automatically saves the script in the project, checks the script into the version control database with version number 1.1.1 and then checks it out so that you can continue working.

The QuickTest status bar indicates each of these operations as they occur. Note, however, that saving your changes to an existing script does not check them in. Even if you save and close the script, it remains checked out until you choose to check it in. For more information, see "Checking Scripts Out of the Version Control Database" on page 233.

## Checking Scripts Out of the Version Control Database

When you choose **File > Open** to open a script that is currently checked in to the version control database, it is opened in read-only mode.

---

**Note:** The Open Test from Quality Center Project dialog box displays icons that indicate the version control status of each script in your project. For more information, see "Opening Scripts from a Quality Center Project" on page 229.

---

You can review the checked-in script. You can also run the script and view the results.

To modify the script, you must check it out. When you check out a script, Quality Center copies the script to your unique check-out directory (automatically created the first time you check out a script), and locks the script in the project database. This prevents other users of the Quality Center project from overwriting any changes you make to the script. However, other users can still run the version that was last checked in to the database.

You can save and close the script, but it remains locked until you return the script to the Quality Center database. You can release the script by either checking the script in, or undoing the check out operation. For more information on checking scripts in, see "Checking Scripts into the Version Control Database" on page 235. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 240.

By default, the check out option accesses the latest version of the script. You can also check out older versions of the script. For more information, see "Using the Version History Dialog Box" on page 237.

**To check out the latest version of a script:**

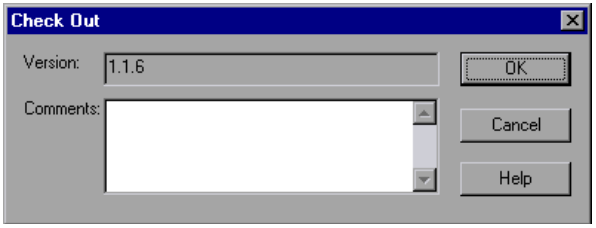
**1** Open the script you want to check out. For more information, see "Opening Scripts from a Quality Center Project" on page 229.

---

**Note:** Make sure the script you open is currently checked in. If you open a script that is checked out to you, the **Check Out** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

---

**2** Choose **File** > **Quality Center Version Control** > **Check Out**. The Check Out dialog box opens and displays the script version to be checked out.




**3** You can enter a description of the changes you plan to make in the **Comments** box.

**4** Click **OK**. The read-only script closes and automatically reopens as a writable script.

## Checking Scripts into the Version Control Database

While a script is checked out, Quality Center users can run the previously checked-in version of your script. For example, suppose you check out version 1.2.3 of a script and make a number of changes to it and save the script. Until you check the script back in to the version control database as version 1.2.4 (or another number that you assign), Quality Center users can continue to run version 1.2.3.

When you have finished making changes to a script and you are ready for Quality Center users to use your new version, you check it in to the version control database.

---

**Note:** If you do not want to check your changes into the Quality Center database, you can undo the check-out operation. For more information, see "Canceling a Check-Out Operation" on page 240.

---

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

**To check in the currently open script:**

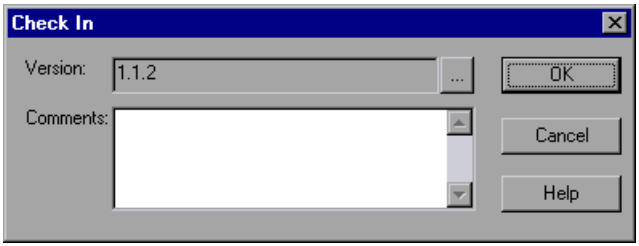
**1** Confirm that the currently open script is checked out to you. For more information, see "Viewing Version Information For a Script" on page 237.

---

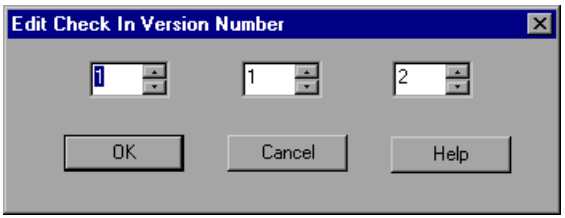
**Note:** If the open script is currently checked in, the **Check In** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

---

**2** Choose **File** > **Quality Center Version Control** > **Check In**. The Check In dialog box opens.




**3** Accept the default new version number and proceed to step 7, or click the browse button to specify a custom version number. If you click the browse button, The Edit Check In Version Number dialog box opens.




**4** Modify the version number manually or using the up and down arrows next to each element of the version number. You can enter numbers 1-900 in the first element. You can enter numbers 1-999 in the second and third elements. You cannot enter a version number lower than the most recent version of this script in the version control database.

**5** Click **OK** to save the version number and close the Edit Check In Version Number dialog box.

**6** If you entered a description of your change when you checked out the script, the description is displayed in the **Comments** box. You can enter or modify the comments in the box.

**7** Click **OK** to check in the script. The script closes and automatically reopens as a read-only script.

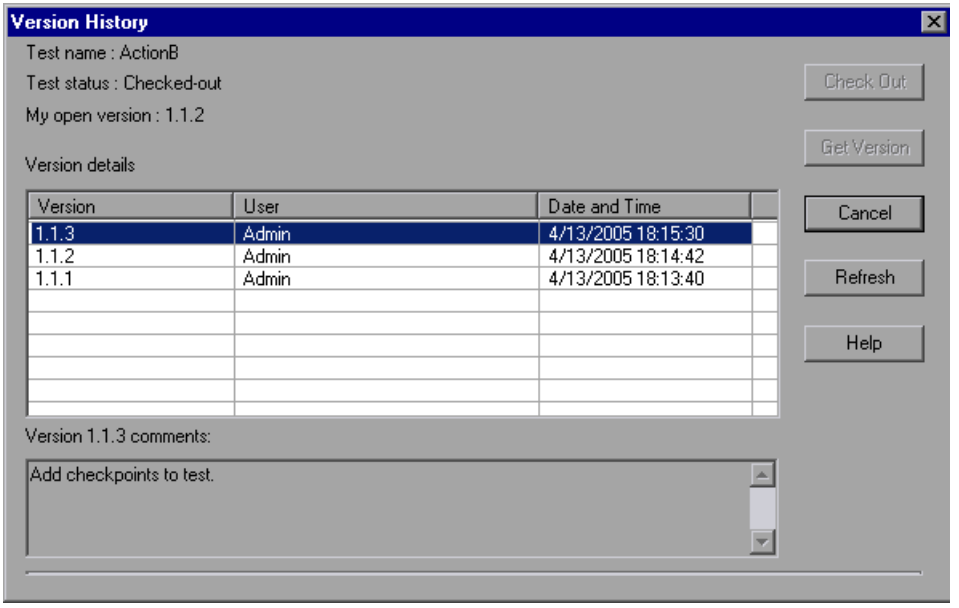
## Using the Version History Dialog Box

You can use the Version History dialog box to view version information about the currently open script and to view or retrieve an older version of the script.

### Viewing Version Information For a Script

You can view version information for any open script that has been stored in the Quality Center version control database, regardless of its current status.

To open the Version History dialog box for a script, open the script and choose **File** > **Quality Center Version Control** > **Version History**.




The Version History dialog box provides the following information:

**Test name**—The name of the currently open script.

**Test status**—The status of the script. The script can be:

- **Checked-in**—The script is currently checked in to the version control database. It is currently open in read-only format. You can check out the script to edit it.
- **Checked-out**—The script is checked out by you. It is currently open in read-write format.
- **Checked-out by <another user>**—The script is currently checked out by another user. It is currently open in read-only format. You cannot check out or edit the script until the specified user checks in the script.

**My open version**—The script version that is currently open on your QuickTest computer.

**Version details—**The version details for the script.

- **Version**—A list of all versions of the script.
- **User**—The user who checked in each listed version.
- **Date and Time**—The date and time that each version was checked in.

**Version comments**—The comments that were entered when the selected version was checked in.

### Working with Previous Script Versions

You can view an old version of a script in read-only mode or you can check out an old version and then check it in as the latest version of the script.

**To view an old version of a script:**

1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 229.
2. Choose **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version you want to view in the **Version details** list.
4. Click the **Get Version** button. QuickTest reminds you that the script will open in read-only mode because it is not checked out.

5 Click **OK** to close the QuickTest message. The selected version opens in read-only mode.

---

**Tips:** To confirm the version number that you now have open in QuickTest, look at the **My open version** value in the Version History dialog box.

After using the **Get Version** option to open an old version in read-only mode, you can check-out the open script by choosing **File** > **Quality Center Version Control** > **Check Out**. This is equivalent to using the **Check Out** button in the Version History dialog box.

---

**To check out an old version of a script:**

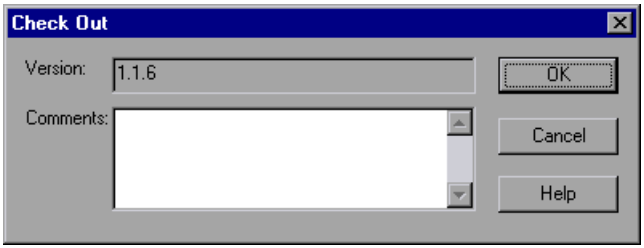
1 Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 229.

2 Choose **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.

3 Select the version you want to view in the **Version details** list.

4 Click the **Check Out** button. A confirmation message opens.

5 Confirm that you want to check out an older version of the script. The Check Out dialog box opens and displays the version to be checked out.




6 You can enter a description of the changes you plan to make in the **Comments** box.

7 Click **OK**. The open script closes and the selected version opens as a writable script.

**8** View or edit the script as necessary.

**9** If you want to check in your script as the new, latest version in the Quality Center database, choose **File** > **Quality Center Version Control** > **Check In**. If you do not want to upload the modified script to Quality Center, choose **File** > **Quality Center Version Control** > **Undo Check out**.

For more information on checking scripts in, see "Checking Scripts into the Version Control Database" on page 235. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 240.

### Canceling a Check-Out Operation

If you check out a script and then decide that you do not want to upload the modified script to Quality Center you should cancel the check-out operation so that the script will be available for check out by other Quality Center users.

**To cancel a check-out operation:**

**1** If it is not already open, open the checked-out script.

**2** Choose **File** > **Quality Center Version Control** > **Undo Check out**.

**3** Click **Yes** to confirm the cancellation of your check-out operation. The check-out operation is cancelled. The checked-out script closes and the previously checked-in version reopens in read-only mode.

# 16

# Managing Scripts with Performance Center

VuGen's integration with Performance Center lets you upload and download scripts to and from the Performance Center server.

This chapter describes:

- About Managing Scripts with Performance Center
- Connecting VuGen to Performance Center
- Uploading Vuser Scripts
- Downloading Vuser Scripts

## About Managing Scripts with Performance Center

VuGen provides integration with Performance Center, Mercury Interactive's Web-enabled global load testing tool that allows you to test your system from different geographical locations.

You can upload and download scripts to and from Performance Center using VuGen's user interface. You upload scripts to Performance Center in order to add them to your Vuser Scripts list and use them in your test. You can also download scripts to edit them or save them locally.

In order for VuGen to access Performance Center for either uploading or downloading, you must connect to the server upon which Performance Center is installed.

For further information about working with Performance Center, refer to the *Performance Center User's Guide*.

# Connecting VuGen to Performance Center

VuGen works together with Performance Center to provide an efficient method for uploading and downloading Vuser scripts to and from Performance Center. In order for VuGen to access a Performance Center project, you must first connect it to the Web server on which Performance Center is installed. You can then upload or download Vuser scripts. For more information, see:

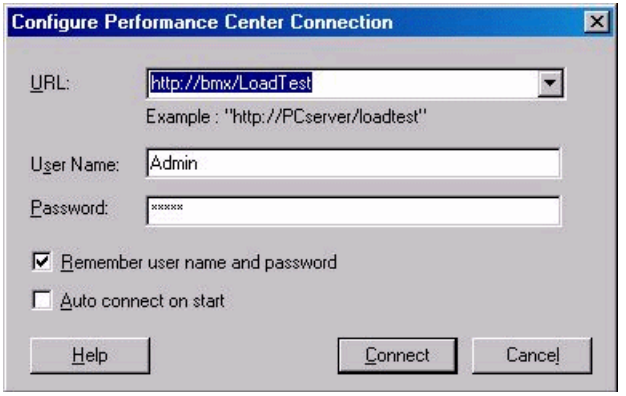
➤ "Uploading Vuser Scripts" on page 244.

➤ "Downloading Vuser Scripts" on page 249.

You connect to Performance Center using the Configure Performance Center Connection dialog box.

**To connect VuGen to Performance Center:**

**1** In VuGen, select **Tools > Configure Performance Center Connection**. The Configure Performance Center Connection dialog box opens.




**2** In the URL box, type the URL address of the Web server on which Performance Center is installed. The URL address should be in the format: http://<server_name>/loadtest

**3** Enter your user name and password. Contact your Performance Center administrator if you need assistance.

4 To automate the login process, select **Remember User Name and Password**. The specified username and password are saved to the registry, and displayed each time you open the dialog box.

5 To automatically open the connection to the Performance Center server when you start VuGen, select **Auto connect when VuGen starts**. VuGen attempts to connect to Performance Center using the configuration information displayed.

6 Click **OK** to connect to Performance Center. The Performance Center Connection dialog box displays the connection status.

Once the connection is established, all the fields are displayed in read-only format.

---

**Note:** If the connection fails, a dialog box displays the reason for the connection failure.

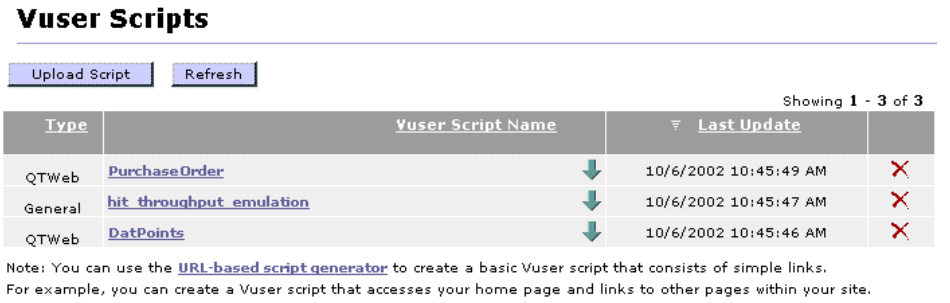
You cannot be connected to Performance Center and Quality Center at the same time.

---

# Uploading Vuser Scripts

In order to use Vuser scripts in your Performance Center project, you need to upload the Vuser scripts to the Performance Center server. The Vuser Scripts list on the Performance Center server displays all the stored Vuser scripts that are available for your project.

To open the Vuser Scripts page, select **Vuser Scripts** from the **Projects** menu.




If you have already added Vuser scripts to the Vuser Scripts list, Performance Center displays them on the Vuser Scripts page. This list represents all the Vuser scripts that are available for use by Vusers during a load test. Scripts uploaded using VuGen are automatically added to the Vuser Scripts list.

If you do not have Vuser scripts in the Vuser Scripts list and you want to add a new script, you can add scripts by:

- Uploading a Vuser Script from VuGen
- Uploading a Vuser Script from Performance Center

## Verifying your Version of VuGen

In order to process the upload, your version of VuGen must be properly configured, and you must connect VuGen to Performance Center.

**To check for proper upload configuration in VuGen:**

1. Start VuGen and open a new or existing VuGen script.
2. Select **Tools**.

If the menu item **Configure Performance Center Connection** is available on the drop-down menu, your version of VuGen is enabled to upload scripts.

If your version of VuGen is not enabled to do uploads or if you do not have VuGen installed on your machine, you need to install a newer version of VuGen. It is recommended that you uninstall your older version. To uninstall VuGen, choose the uninstall option under the Virtual User Generator from the **Start** menu.

**To install a newer version of VuGen:**

1 From the **Miscellaneous** menu, select **Downloads**.

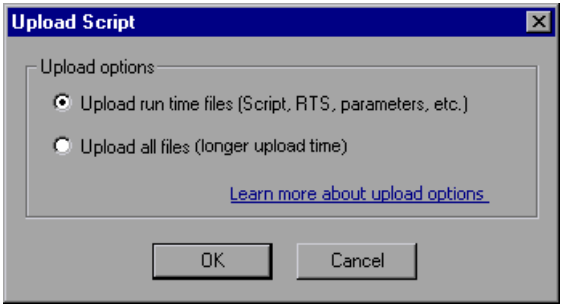
2 Select **Standalone LoadRunner VuGen**.

3 Follow the download instructions.

Once you have a version of VuGen that is enabled, you can upload existing scripts or record new scripts to upload.

## Upload Options

You can upload Vuser scripts from within VuGen to the Performance Center Web site script repository. Depending on your requirements, you can do a partial or a complete upload. The upload options are:




➤ **Upload run time files:** First VuGen deletes all main script files (*usr*, *c*, *cfg*, and *xml* files) from the server. It does not delete data files or old recorded data. Next, VuGen uploads the script files, the run-time settings, and the parameter files.

➤ **Upload all files:** First VuGen deletes **all** script and data files from the server. VuGen then uploads the current script and data files, including the recording data and the replay result directories.

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

---

**Note:** If you previously downloaded script files, VuGen by default uploads only the files that were downloaded. If you want to upload newly created files, for example, you replayed the downloaded script to create snapshots, you must specify that all files are uploaded.

---

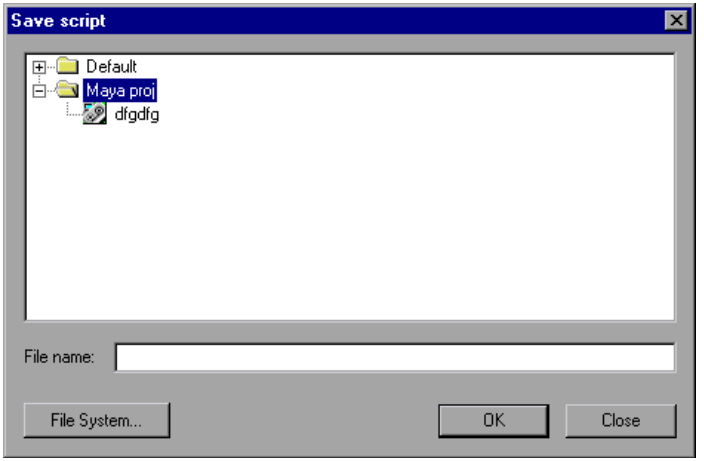
### Uploading a Vuser Script from VuGen

Once you are connected to VuGen, you can upload your script files to the Performance Center server. For more information, see "Connecting VuGen to Performance Center," on page 242.

If VuGen is not connected to Performance Center, you can save the Vuser script files locally to the file system. Later, when VuGen is connected to Performance Center, open the script in VuGen and upload it to Performance Center as described below.

**To upload a Vuser script to Performance Center from VuGen:**

**1** Choose **File** > **Save** in VuGen. The Save Script dialog box opens.




**2** Select the project where you want to save the script. Type name for the script in the **File name** box.

---

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

---

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

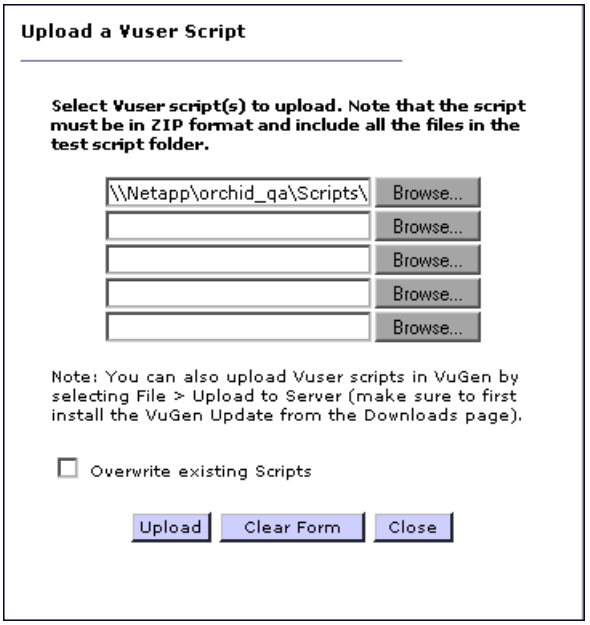
**4** Click **OK** to upload the files to the Performance Center server.

## Uploading a Vuser Script from Performance Center

If your installation does not include VuGen, you can still upload scripts using the Upload Script function in Performance Center's **Vuser Scripts** page.

**To upload a Vuser script from Performance Center:**

1. Open the Vuser Scripts page.
2. Click **Upload Script**. The Upload a Vuser Script dialog box opens.




3. Click a **Browse** button to browse to the zip file containing each Vuser script you want to upload. Note that the zip file must contain the complete contents of the Vuser script folder, including the Vuser script file (".usr" file) itself and all related data files.
4. Select the zip file, and click **Open**.
5. Check **Overwrite existing Scripts** if you want to replace a script that already exists in the Vuser Scripts list.
6. Click **Upload** to upload the script and add it to your Vuser Scripts list.

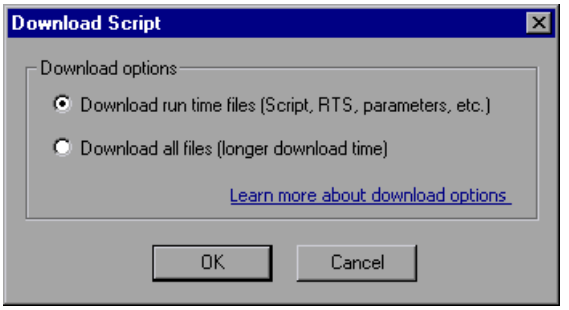
# Downloading Vuser Scripts

VuGen works together with Performance Center to provide an efficient method for downloading Vuser scripts from Performance Center for editing, and automatically opening them in VuGen. You can choose to download only the script run time files, or the complete files including the recording data and replay results.

In order for VuGen to access a Performance Center project, you must first connect it to the Web server on which Performance Center is installed. You can then select the script files that you want to download. You connect to Performance Center from the Configure Performance Center Connection dialog box. For more information, see "Connecting VuGen to Performance Center," on page 242.

## Download Options

You can download Vuser scripts from the Performance Center script repository to VuGen. Depending on your requirements, you can do a partial or a complete download. The download options are:




- **Download run time files:** VuGen downloads the script files only, allowing faster downloads. This includes the script file, run-time settings, and parameter files.
- **Download all files:** VuGen downloads the script and data files, including the recording data and the replay result directories.

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

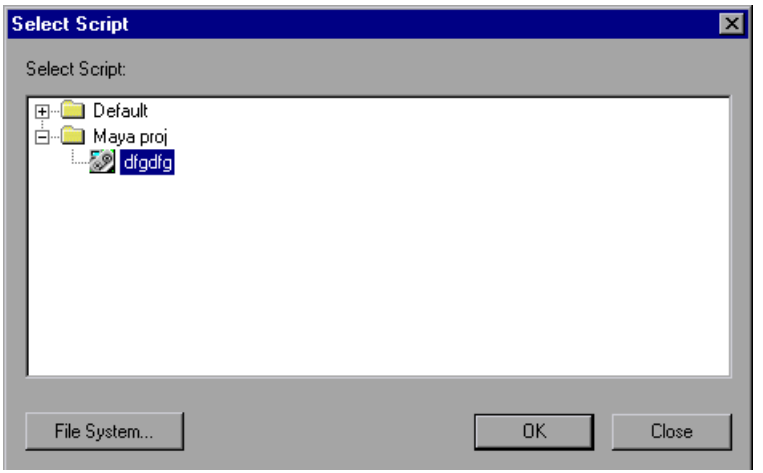
## Downloading a Vuser Script from VuGen

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

**To download a Vuser script from Performance Center:**

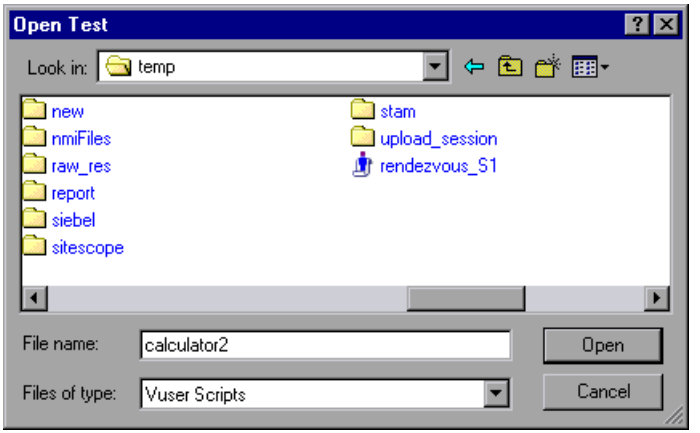
1 Connect to the Performance Center server. For more information, see "Connecting VuGen to Performance Center" on page 242.

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



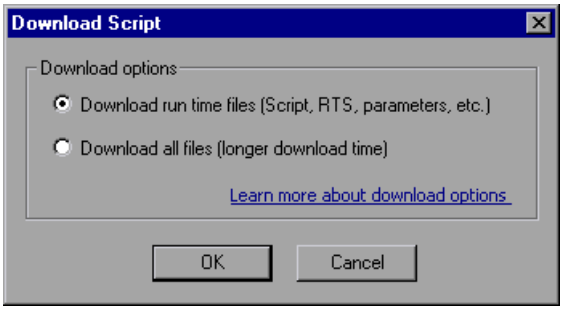

3 Select the script that you want to download.

To select a script from a local drive (even if you are connected to Performance Center), click **File System**. The Open Test dialog box opens.




Browse to the file that you want to download, and click **Open**. The Performance Center Select Script dialog box reopens. Select the script that you want to download.

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




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

**6** Click **OK** to download the files from Performance Center. When the download is complete, the dialog box closes and VuGen displays the script.

By default, downloaded files are saved to your temp directory. To save them to a different directory, click **Save** and specify a directory.

# Part III

# 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 Vusers.

# 17

# 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:

- About Recording Java Language Vuser Scripts
- 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 Vuser API Java-specific functions. After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it. Once you verify that the script is functional, you incorporate it into a LoadRunner scenario, Tuning Module session step, or Business Process Monitor profile.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. Refer to Chapter 29, "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 VuGen.

VuGen provides a tool that enables you to convert a Vuser script created for Web, into Java. For more information, see "Converting Web Vuser Scripts into Java" on page 544.

## 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 29, "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 18, "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 Vuser API specific functions to enhance the Vuser script. For details, see Chapter 29, "Programming Java Scripts." You can use the built-in Java function Navigator. For more information, see "Viewing the Java Methods" on page 259.

**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 8, "Working with VuGen 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 20, "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 14, "Running Vuser Scripts in Standalone Mode."

For detailed information on the recording procedure, refer to the specific chapter for your Vuser type.

## Understanding Java Language Vuser Scripts

When you record a session, VuGen logs all calls to the server and generates a script with functions. These functions describe all of your actions within the application or applet. The script also contains supplementary code required for proper playback, such as property settings, and naming service initialization (JNDI).

The recorded script is comprised of three sections:

- Imports
- Code
- Variables

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script. The **Code** section contains the Actions class and the recorded code within the **init**, **actions**, and **end** methods. The **Variables** section, after the **end** method, contains all the type declarations for the variables used in the code.

After you finish recording, you can modify the functions in your script, or add additional Java or Mercury 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, refer to 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 443.

## 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.

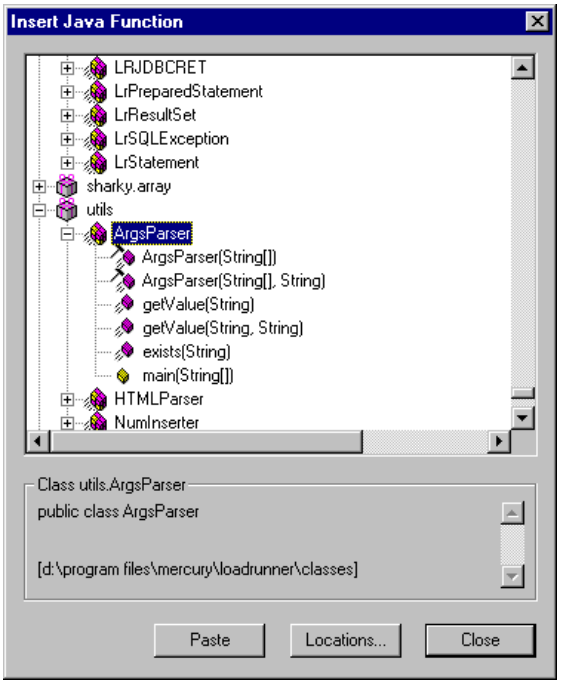
To use the protected methods, add the Vuser to the package of required methods. At the beginning of your script, add the following line:

```
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 261.

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 () |

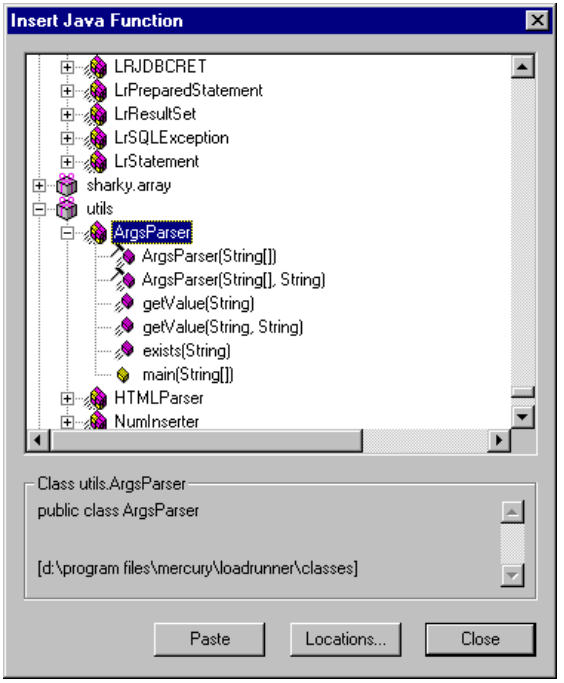
# Manually Inserting Java Methods

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 263.

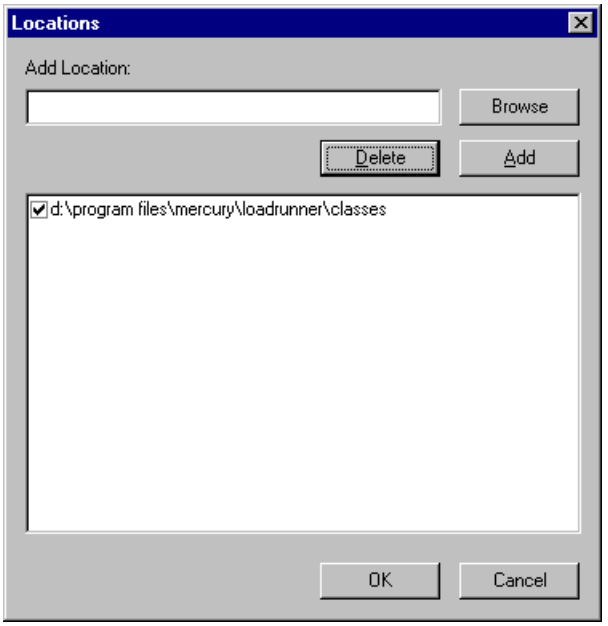
**To insert Java functions:**

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 setting **DefaultValues** is set to **true**, you can use the default values inserted by VuGen. If **DefaultValues** is set to **false**, you must add parameters for all methods you insert into the script.

In addition, modify any return values. For example, if your script generated the following statement "(String)=LavaVersion.getVersionId();", replace (String) with a string type variable.

**13** Add any necessary statements to your script such as imports or Vuser API Java functions (described in Chapter 29, "Programming Java Scripts").

**14** Save the script and run it from VuGen.

## Configuring Script Generation Settings

You can customize the way the navigator adds methods to your script in the following areas:

- Class Name Path
- Automatic Transactions
- Default Parameter Values
- Class Pasting

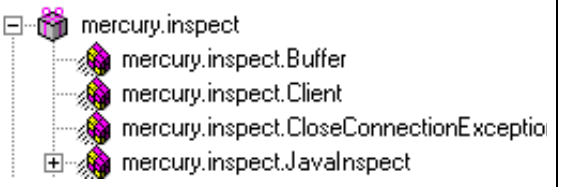
To view the configuration setting, open the **jquery.ini** file in VuGen's dat directory.

```
[Display]
FullClassName=False

[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

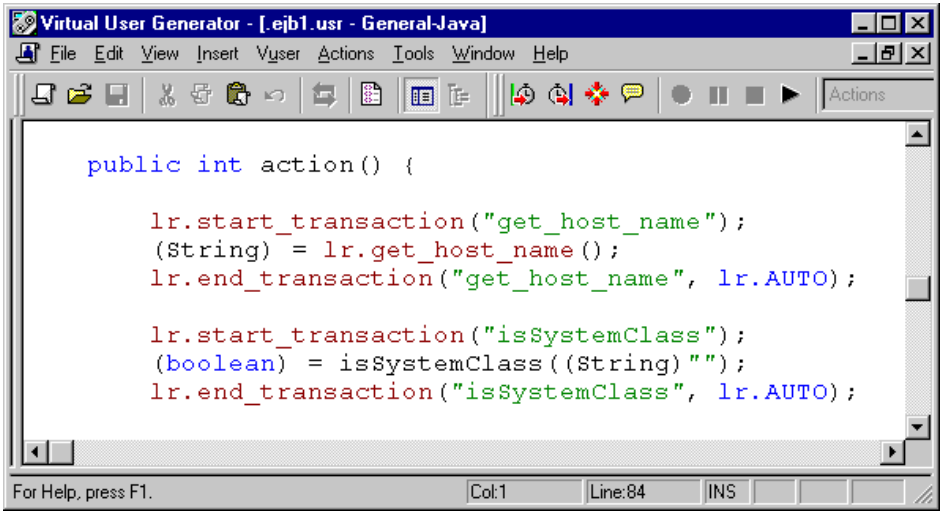
### Class Name Path

The **FullClassName** option displays the complete package and class name in the Java Function navigator. This option does not affect the way the functions are added into the script—it only affects the way the classes are displayed in the navigator. By default, this option is set to false. If your packages have many classes and you are unable to view the package and class names at the same time, you should enable this option.

| FullClassName enabled | FullClassName disabled |
|---|---|
|  |  |

## Automatic Transactions

The **AutoTransaction** setting creates a Vuser transaction for all methods. When you enable this option, VuGen automatically encloses all Java methods with **lr.start_transaction** and **lr.end_transaction** functions. This allows you to individually track the performance of each method. This option is disabled by default.




## Default Parameter Values

The **DefaultValues** setting includes default values for all methods you paste into your script. This option is enabled by default and inserts a null for all objects. If you disable this option, you must manually insert parameter values for all functions in the script. The following table illustrates the **DefaultValues** flag enabled and disabled.

| DefaultValues enabled | DefaultValues disabled |
|---|---|
| lr.message((String)""); | lr.message((String)); |
| lr.think_time((int)0); | lr.think_time((int)); |
| lr.enable_redirection((boolean)false); | lr.enable_redirection((boolean)); |
| lr.save_data((byte[])null, (String)""); | lr.save_data((byte[]), (String)); |

## Class Pasting

The **CleanClassPaste** setting pastes a class so that it will compile cleanly: with an instance returning from the constructor, with default values as parameters, and without a need for import statements. Using this option, you will most likely be able to run your script without any further modifications. If you disable this option (default), you may need to manually define parameters and include import statements. Note that this setting is only effective when you paste an entire class into your script—not when you paste a single method.

The following segment shows the **toString** method pasted into the script with the **CleanClassPaste** option enabled.

```
_class.toString();
    // Returns: java.lang.String
```

The same method with the **CleanClassPaste** option disabled is pasted as follows:

```
(String) = toString();
```

The next segment shows the **NumInserter** Constructor method pasted into the script with the **CleanClassPaste** option enabled.

```
utils.NumInserter _numinserter = new utils.NumInserter
            ((java.lang.String)"", (java.lang.String)"", (java.lang.String)""…);
// Returns: void
```

The same method with the **CleanClassPaste** option disabled is pasted as:

```
new utils.NumInserter((String)"", (String)"", (String)"",...);
```

# 18

# 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:

- About Setting Java Recording Options
- Java Virtual Machine (JVM) Recording Options
- Setting Classpath Recording Options
- Recorder Options
- Serialization Options
- Correlation Options
- Debug Options
- CORBA 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 55, "Performing EJB Testing."

Before recording, VuGen lets you set recording options for the Java Virtual Machine (JVM) and for the code generation stage. Setting the recording options is not mandatory; if you do not set them, VuGen uses the default values.

The options described in this chapter were previously handled by modifying the **mercury.properties** file.

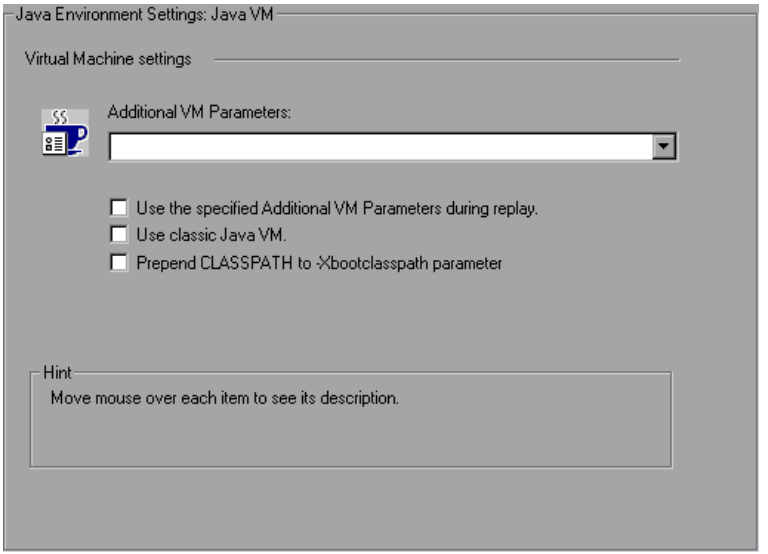
You can set recording options in the following areas:

- Java Virtual Machine (JVM) Recording Options
- Setting Classpath Recording Options
- Recorder Options
- Serialization Options
- Correlation Options
- Debug Options

# Java Virtual Machine (JVM) Recording Options

The **Java VM** options indicate additional parameters to use when recording Java applications.

When you record a Vuser, VuGen automatically sets the **Xbootclasspath** variable with default parameters. If you use this dialog box to set the **Xbootclasspath** with different parameters, it will use those command parameters—not the default ones.




You can also instruct VuGen to add the Classpath before the **Xbootclasspath** (prepend the string) to create a single Classpath string.

By default, VuGen uses the classic VM during recording. You can also instruct VuGen to use another virtual machine (Sun's Java Hotspot VM).

**To set the Java Virtual Machine recording options:**

1. Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Java VM** node in the Recording Options tree.

2. In the **Additional VM Parameters** box, list the Java command line parameters. These parameters may be any Java VM argument. The common arguments are the debug flag (**-verbose**) or memory settings (**-ms**, **-mx**). For more information about the Java VM flags, see the JVM documentation. In additional, you may also pass properties to Java applications in the form of a **-D** flag.

   VuGen automatically sets the **-Xbootclasspath** variable (for JDK 1.2 and higher) with default parameters. When you specify **-Xbootclasspath** with parameter values as an additional parameter, VuGen uses this setting instead of the default one.

3. To use the same Additional VM parameters in replay, select the **Use the specified Additional VM Parameters during replay** check box.

4. To use the classic VM, select the **Use classic Java VM** check box (default). To use another VM (Sun's Java HotSpot), clear the check box.

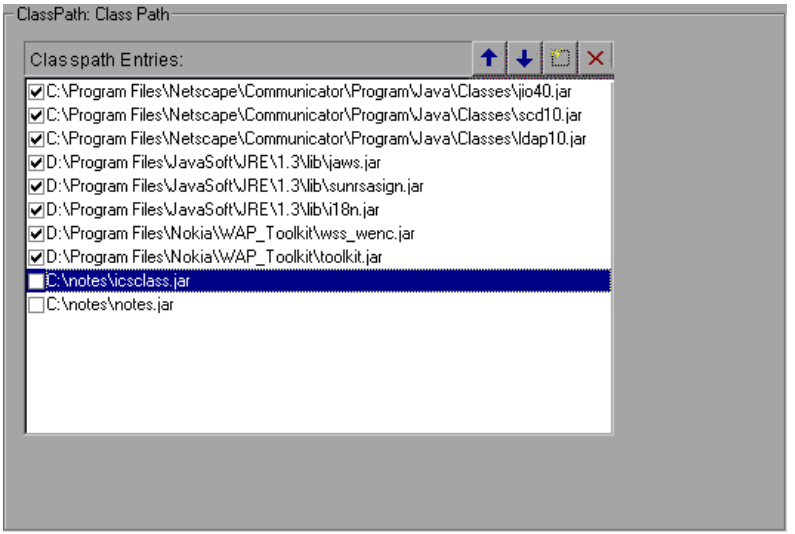
5. To add the Classpath before the **Xbootclasspath** (prepend the string), select the **Prepend CLASSPATH to -Xbootclasspath parameter** check box.

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

# Setting Classpath Recording Options

The **Java Environment Settings**:**Classpath** node lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper recording.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.




**To set the Classpath recording options:**

1 Click **Options** in the Start Recording dialog box. Select the **Java Environment Settings:Classpath** node in the Recording Options tree.

2 To add a classpath to the list:



Click the **Add Classpath** button. VuGen adds a new line to the classpath list.

Type in the path and name of the **jar**, **zip** or other archive file for your class. Alternatively, click the **Browse** button to the right of the field, and locate the desired file. VuGen adds the new location to the classpath list, with an enabled status.



3 To permanently remove an entry, select it and click the Delete button.

4 To disable a classpath entry for a specific test, clear the check box to the left of the entry.



5 To move an entry down in the list, select it and click the Down arrow.



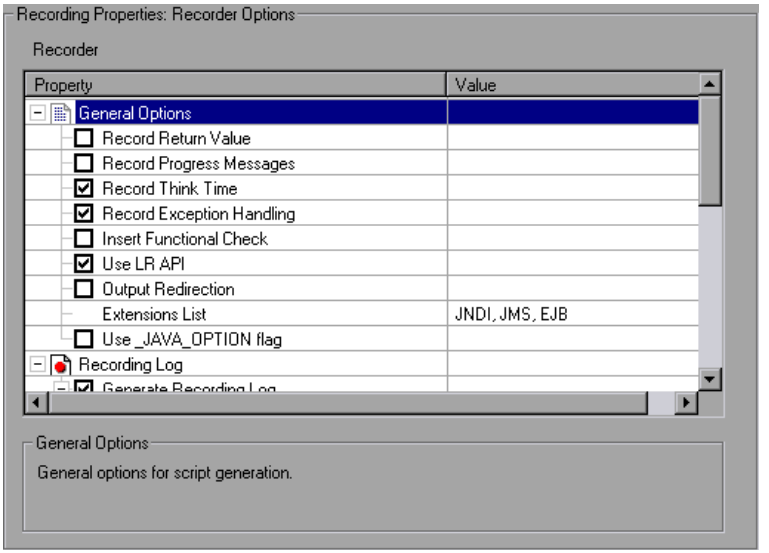
6 To move a classpath entry up within the list, select it and click the Up arrow.

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

## Recorder Options

The **Recorder** options 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 (enabled 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 testing or 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 (disabled by default).

**Unreadable Strings as Bytes:** Represents strings containing unreadable characters as byte arrays. This option applies to strings that are passed as parameters to invocations (enabled by default).

**Byte Array Format:** The format of byte arrays in a script: **Regular**, **Unfolded Serialized Objects**, or **Folded Serialized Objects**. Use one of the serialized object options when recording very long byte arrays. The default is **Regular**.

**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 290. The following options are available:

**Unfold Serialized Objects:** Expands serialized objects in ASCII representation. This option allows you to view the ASCII values of the objects in order to perform parameterization (enabled by default).

**Limit Object Size (bytes):** Limits serializable objects to the specified value. Objects whose size exceeds this value, will not be given ASCII representation in the script. The default value is 3072.

**Ignore Serialized Objects:** Lists the serialized objects not to be unfolded when encountered in the recorded script. Separate objects with commas.

**Serialization Delimiter:** Indicates the delimiter separating the elements in the ASCII representation of objects. VuGen will only parameterize strings contained within these delimiters. The default delimiter is '#'.

**Unfold Arrays:** Expands array elements of serialized objects in ASCII representation. If you disable this option and an object contains an array, the object will not be expanded. By default, this option is enabled—all deserialized objects are totally unfolded.

**Limit Array Entries:** Instructs the recorder not to open arrays with more than the specified number of elements. The default value is 200.

**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).

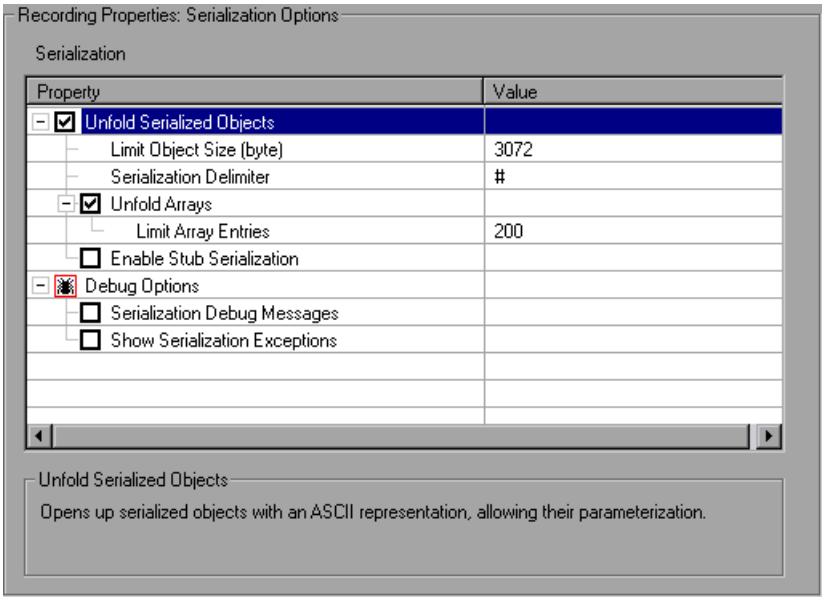
### Debug Options

**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. To set all options to their default values, click **Use Defaults**.

3 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 19, "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).

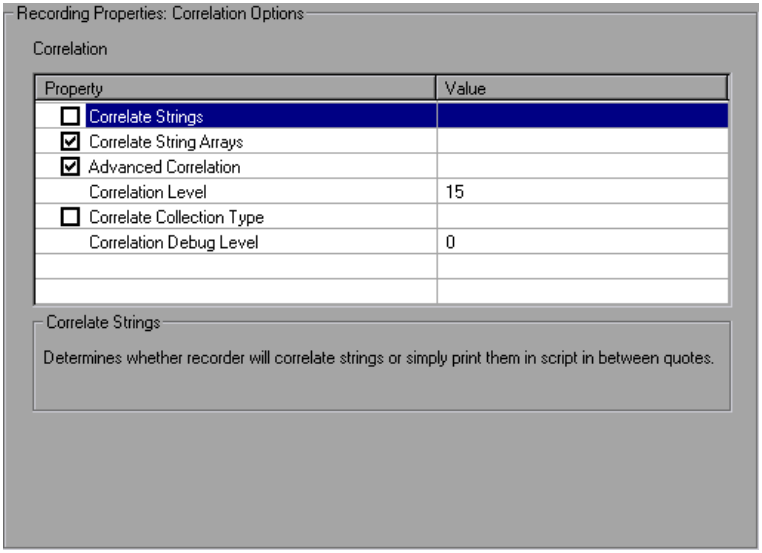
**Correlation Level:** Indicates the level of deep correlation, the number of inner containers to be scanned (15 by default).

**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. To set all options to their default values, click **Use Defaults**.

3 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 option 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 **trace.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.

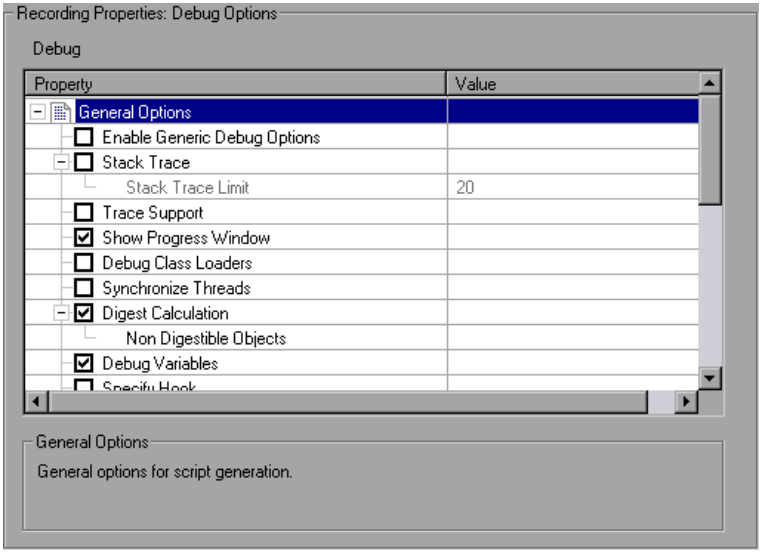
**Dump Suffix:** The suffix to append to the dumped class names. The default suffix is _DUMP.

**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.

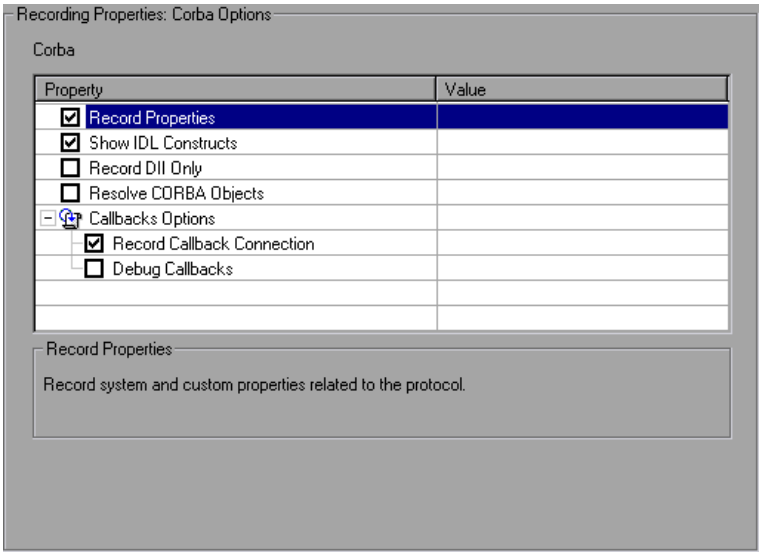
### Callback Options

**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.

# 19

# 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:

- About Correlating Java Scripts
- 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-Java or RMI-Java Vuser script, the dynamic data is recorded into scripts, but cannot be re-used during replay. If you encounter an error when running your Vuser, examine the script at the point where the error occurred. In many cases, correlation will solve the problem by enabling you to use the results of one statement as input to another.

VuGen's 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 290.

## Standard Correlation

Standard correlation refers to the automatic correlation performed during recording for simple objects, excluding object arrays, vectors, and container constructs.

When the recorded application invokes a method that returns an object, VuGen's correlation mechanism records these objects. When you run the script, VuGen compares the generated objects to the recorded objects. If the objects match, the same object is used. The following example shows two CORBA objects **my_bank** and **my_account**. The first object, my_bank, is invoked; the second object, my_account, is correlated and passed as a parameter in final line of the segment:

```
public class Actions {

  // Public function: init
      public int init() throws Throwable {

       Bank my_bank = bankHelper.bind("bank", "shunra");
       Account my_account = accountHelper.bind("account","shunra");

       my_bank.remove_account(my_account);
      }
:
}
```

## Advanced Correlation

Advanced or **deep** correlation refers to the automatic correlation performed during recording for complex objects, such as object arrays and CORBA container constructs.

The deep correlation mechanism handles CORBA constructs (structures, unions, sequences, arrays, holders, 'any's) as containers. This allows it to reference inner members of containers, additional objects, or different containers. Whenever an object is invoked or passed as a parameter, it is also compared against the inner members of the containers.

In the following example, VuGen performs deep correlation by referencing an element of an array. The **remove_account** object receives an account object as a parameter. During recording, the correlation mechanism searches the returned array my_accounts and determines that its sixth element should be passed as a parameter.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_banks[] = bankHelper.bind("banks", "shunra");
    my_accounts[] = accountHelper.bind("accounts","shunra");

    my_banks[2].remove_account(my_accounts[6]);
  }
:
}
```

The following segment further illustrates enhanced correlation. The script invokes the **send_letter** object that received an **address** type argument. The correlation mechanism retrieves the inner member, address, in the sixth element of the my_accounts array.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_banks = bankHelper.bind("bank", "shunra");
    my_accounts = accountHelper.bind("account", "shunra");

    my_banks[2].send_letter(my_accounts[6].address);
  }
:
}
```

## String Correlation

String correlation refers to the representation of a recorded value as an actual string or a variable. When you disable string correlation (the default setting), the actual recorded value of the string is indicated explicitly within the script. When you enable string correlation, it creates a variable for each string, allowing you to use it at a later point in the script.

In the following segment, string correlation is enabled—you store the value returned from the **get_id** method in a string type variable for use later on in the script.

```
public class Actions {

  // Public function: init
  public int init() throws Throwable {

    my_bank = bankHelper.bind("bank", "shunra");
    my_account1 = accountHelper.bind("account1", "shunra");
    my_account2 = accountHelper.bind("account2", "shunra");

    string = my_account1.get_id();
    string2 = my_account2.get_id();
    my_bank.transfer_money(string, string2);
  }
:
}
```

You set the correlation method from the **Correlation** tab in the recording options.

**Correlate Strings:** Correlate strings in script during recording. If you disable this option, the actual recorded values are included in the script between quotation marks. If this option is disabled, all other correlation options are ignored (disabled by default).

**Correlate String Arrays:** Correlate strings within string arrays during recording. If you disable this option, strings within arrays are not correlated and the actual values are placed in the script (enabled by default).

**Advanced Correlation:** Enables correlation on complex objects such as arrays and CORBA container constructs and arrays. This type of correlation is also known as deep correlation (enabled by default).

**Correlation Level:** Determines the level of deep correlation—how many inner containers to search.

**Correlate Collection Type:** Correlate objects contained in a Collection class for JDK 1.2 or higher (disabled by default).

## Using the Serialization Mechanism

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

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

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user directory. Additional parameters are saved as new binary data files, numbered sequentially. VuGen generates the following code:

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
        map.set_point(p);
     }
:
}
```

The integer passed to **lr.deserialize** represents the number of binary data files in the Vuser directory.

To parameterize the recorded value, use the public **setLocation** method (for information, see the JDK function reference). The following example uses the **setLocation** method to set the value of the object, **p**.

```
public class Actions {

  // Public function: init
      public int init() throws Throwable {
          java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
          p.setLocation(2,9);
          map.set_point(p);
      }
:
:
}
```

In certain instances the public method of **setLocation** is not applicable. As an alternative, you can use the API of your class that incorporate get or set accessor methods. If you are working with AUT classes that do not have get/set methods or use private methods, or if you are unfamiliar with the classes' API, you can use VuGen's built-in serialization mechanism. This mechanism allows you to expand objects in their ASCII representation and manually parameterize the script. You enable this mechanism in the Recording Options dialog box (see Chapter 18, "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 index number of binary file to load.

If you choose not to expand objects in your script by clearing the **Unfold Serialized Objects** check box, you can control the serialization mechanism by passing arguments to the **lr.deserialize** method. The first argument is an integer indicating the number of binary files to load. The second integer is a boolean value:

| | |
|---|---|
| **true** | Use VuGen's serialization mechanism. |
| **false** | Use the standard Java serialization mechanism. |

The following segment shows a generated script in which the serialization mechanism was enabled.

```
public class Actions {

  // Public function: init
     public int init() throws Throwable {
        _string = "java.awt.Point __CURRENT_OBJECT = {" +
                  "int x = "#5#" +
                  "int y = "#8#" +
        "}";
        java.awt.Point p = (java.awt.Point)lr.deserialize(_string,0);
        map.set_point(p);
     }
:
}
```

The string values are placed between delimiters.The default delimiter is "#". You can change the delimiter in the **Serialization** tab of the recording options. Delimiters are used to speed up the parsing of the string during replay.

When modifying the string, you must maintain the following rules:

- Order of lines may not be changed. The parser reads the values one-by-one—not the member names.
- Only values between two delimiters may be modified.
- Object references may not be modified. Object references are indicated only to maintain internal consistency.

- "_NULL_" can appear as a value, representing the Java null constant. You can replace it with string type values only.
- Objects may be deserialized anywhere in the script. For example, you can deserialize all objects in the **init** method and use the values in the **action** method.
- Maintain internal consistency for the objects. For example, if a member of a vector is **element count** and you add an element, you must modify the element count.

  In the following segment, a vector contains two elements:

```
public class Actions {

 // Public function: init
    public int init() throws Throwable {
        _string = "java.util.Vector CURRENTOBJECT = {" +
                  "int capacityIncrement = "#0#" +
                  "int elementCount = #2#" +
                  "java/lang/Object elementData[] = {" +
                        "elementData[0] = #First Element#" +
                        "elementData[1] = #Second Element#" +
                        "elementData[2] = _NULL_" +
                         ....
                        "elementData[9] = _NULL_" +

                  "}" +
        "}";
        _vector = (java.util.Vector)lr.deserialize(_string,0);
        map.set_vector(_vector);
    }
:
}
```

In the following example, one of the vector's elements was changed—a "_NULL_" value was changed to "Third element". In coordination with the addition of the new element, the "elementCount" member was modified to "3".

```
public class Actions {

 // Public function: init
    public int init() throws Throwable {
        _string = "java.util.Vector CURRENTOBJECT = {" +
                  "int capacityIncrement = "#0#" +
                  "int elementCount = #3#" +
                  "java/lang/Object elementData[] = {" +
                        "elementData[0] = #First Element#" +
                        "elementData[1] = #Second Element#" +
                        "elementData[2] = #Third Element#" +
                          ....
                        "elementData[9] = _NULL_" +
                  "}" +
        "}";
        _vector = (java.util.Vector)lr.deserialize(_string,0);
        map.set_vector(_vector);
    }
:
}
```

Due to the complexity of the serialization mechanism, which opens up the objects to ASCII representation, opening large objects while recording may increase the time required for script generation. To decrease this time, you can specify flags which will improve the performance of the serialization mechanism.

When adding **lr.deserialize** to your script, 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 will deserialize strings for every iteration.

The following list shows the available options which you set in **Serialization** tab of the recording options:

- Serialization Delimiter
- Unfold Serialized Objects
- Unfold Arrays
- Limit Array Entries
- Ignore Serialized Objects

For complete information on the recording options, see Chapter 18, “Setting Java Recording Options.”

# 20

# 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:

- About Configuring Java Run-Time Settings
- Specifying the JVM Run-Time Settings
- Setting the Run-Time Classpath Options

***The following information applies to Java, EJB Testing, Corba-Java, and RMI-Java type Vusers.***

## About Configuring 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.

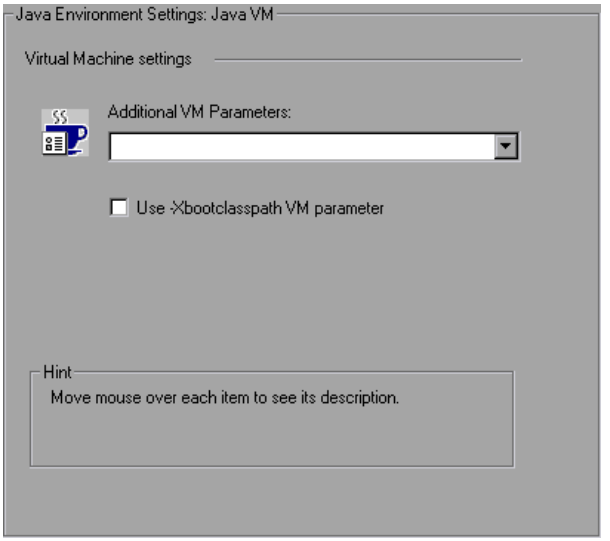


To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar.

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 12, "Configuring Run-Time Settings."

# Specifying the JVM Run-Time Settings

In the Java VM section, you provide information about the Java virtual machine settings.




The following settings are available:

**Additional VM Parameters:** Enter any optional parameters used by the virtual machine.

**Use Xbootclasspath VM parameter:** 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.

**To set the Java VM run-time settings:**

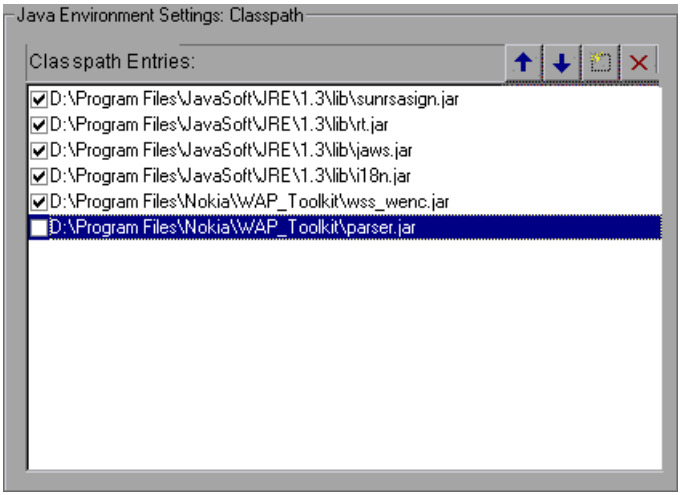
1. Choose **Vuser** > **Run-Time Settings** and select the **Java Environment Settings:Java VM** node in the Run-Time Settings tree.
2. In the **Additional VM Parameters** box, enter any optional parameters used by the Load Generator machine.
3. To replay with the **-Xbootclasspath/p** option, select the **Use -Xbootclasspath VM parameter** option.

**4** Click **OK**.

## Setting the Run-Time Classpath Options

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper replay.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.




**To set the Classpath run-time settings:**

**1** Open the Run-Time settings (F4). Select the **Java Environment Settings:Classpath** node in the Run-Time settings tree.

**2** Add a classpath to the list:



Click the **Add Classpath** button. VuGen adds a new line to the classpath list.

Type in the path and name of the **jar**, **zip** or other archive file for your class. Alternatively, click the **Browse** button to the right of the field, and locate the desired file. VuGen adds the new location to the classpath list, with an enabled status.



**3** To permanently remove a classpath entry, select it and click the Delete button.

**4** To disable a classpath entry for a specific test, clear the check box to the left of the entry.

**5** To move a classpath entry down in the list, select it and click the Down arrow.





**6** To move a classpath entry up within the list, select it and click the Up arrow.

**7** Click **OK** to close the dialog box.

# Part IV

## Application Deployment Solution Protocols

# 21

# Creating 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.

The optional **Mercury Citrix Agent** helps you create an intuitive script that provides built-in synchronization. For more information, see Chapter 22, "Using the LoadRunner Citrix Agent." Make sure to refer to "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 333 for valuable tips on creating scripts.

This chapter describes:

- Getting Started with Citrix Vuser Scripts
- Setting Up the Client and Server
- Recording Tips
- Understanding Citrix Recording Options
- Setting the Citrix Recording Options
- Setting the Citrix Display Settings
- Setting the Citrix Run-Time Settings
- Viewing and Modifying Citrix Vuser Scripts
- Synchronizing Replay
- Understanding ICA Files
- Using Citrix Functions
- Tips for Replaying and Troubleshooting Citrix Vuser Scripts
- Increasing the Numbers of Vusers per Load Generator Machine

# About Creating Citrix Vuser Scripts

Citrix Vuser scripts emulate the Citrix ICA protocol communication between a Citrix client and server. VuGen records all activity during the communication and creates a Vuser script.

When you perform actions on the remote server, VuGen generates functions that describe these actions. Each function begins with a **ctrx** prefix. These functions emulate the analog movements of the mouse and keyboard. In addition, the **ctrx** functions allow you to synchronize the replay of the actions, by waiting for specific windows to open.

VuGen also allows you to record a Citrix NFUSE session. With Citrix NFuse, the client is installed, but your interface is a browser instead of a client interface. To record NFUSE sessions, you must perform a multi-protocol recording for Citrix and Web Vusers. (See Chapter 4, "Recording with VuGen.") In multi-protocol mode VuGen generates functions from both Citrix and Web protocols during recording.

In the following example, **ctrx_mouse_click** simulates a mouse click on the left button.

```
ctrx_mouse_click(44, 318, LEFT_BUTTON, 0);
```

For more information about the syntax and parameters, refer to the *Online Function Reference* (**Help** > **Function Reference**).

You can view and edit the recorded script from VuGen's main window. The API calls that were recorded during the session are displayed in VuGen, allowing you to track your actions.

# Getting Started with Citrix Vuser Scripts

This section provides an overview of the process of developing Citrix ICA Vuser scripts using VuGen. In addition, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 333.

**To develop a Citrix ICA script:**

1 **Make sure that your client and server are configured properly.**

For general information about these settings, see "Setting Up the Client and Server" on page 306.

2 **Record the actions using VuGen.**

Invoke VuGen and create a new Vuser script. Be sure to follow the "Setting Up the Client and Server" on page 306.

For general information about recording, see Chapter 4, "Recording with VuGen."

3 **Enhance the Vuser script.**

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 7, "Enhancing Vuser Scripts."

4 **Define parameters (optional).**

Define parameters for the fixed-values recorded into your Vuser script. By substituting fixed-values with parameters, you can repeat the same business process many times using different values.

For details, see Chapter 8, "Working with VuGen Parameters."

5 **Configure the Citrix display options.**

Configure the display options for replaying Citrix Vusers. These options let you show the Citrix client during replay and open a snapshot when an error occurs. For details, see "Setting the Citrix Display Settings" on page 317.

6 **Configure the Run-Time settings.**

The Run-Time settings control Vuser behavior during script execution. These settings include pacing, logging, think time, and connection information.

For details about the Citrix specific Run-Time settings, see "Setting the Citrix Run-Time Settings" on page 318. For information about general Run-Time settings, see Chapter 12, "Configuring Run-Time Settings."

**7 Save and run the Vuser script from VuGen.**

Save and run the Vuser script from VuGen to verify that it runs correctly. While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script.

For details about running the Vuser script as a standalone test, see "Tips for Replaying and Troubleshooting Citrix Vuser Scripts" on page 333 and Chapter 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Setting Up the Client and Server

Before creating a script, make sure you have a supported Citrix client installed on your machine, and that your server is properly configured. This section describes:

- Client Versions
- Server Configuration

### Client Versions

In order to run your script, you must install a Citrix client on each Load Generator machine. If you do not have a client installed, you can download one from the Citrix Website www.citrix.com under the **download** section.

VuGen supports all Citrix clients with the exception of versions 8.00, version 6.30.1060 and earlier, and Citrix Web clients.

## Server Configuration

To record in VuGen, you need to configure the Citrix server in the following areas:

**MetaFrame:** Make sure the MetaFrame server (1.8 or 3) is installed. To check the version of the server, select **Citrix Connection Configuration** on the server's console toolbar and choose **Help** > **About**.

**Configure Server to Close Sessions:** Configure the Citrix server to completely close a session. After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time that client opens a new connection. Consequently, a new connection by the same client will face the same workspace from which it disconnected previously. It is preferable to allow each new test run to use a clean workspace.

The ensure a clean workspace for each test, you must configure the Citrix server not to save the previous session. Instead, it should reset the connection by disconnecting from the client each time the client times-out or breaks the connection.

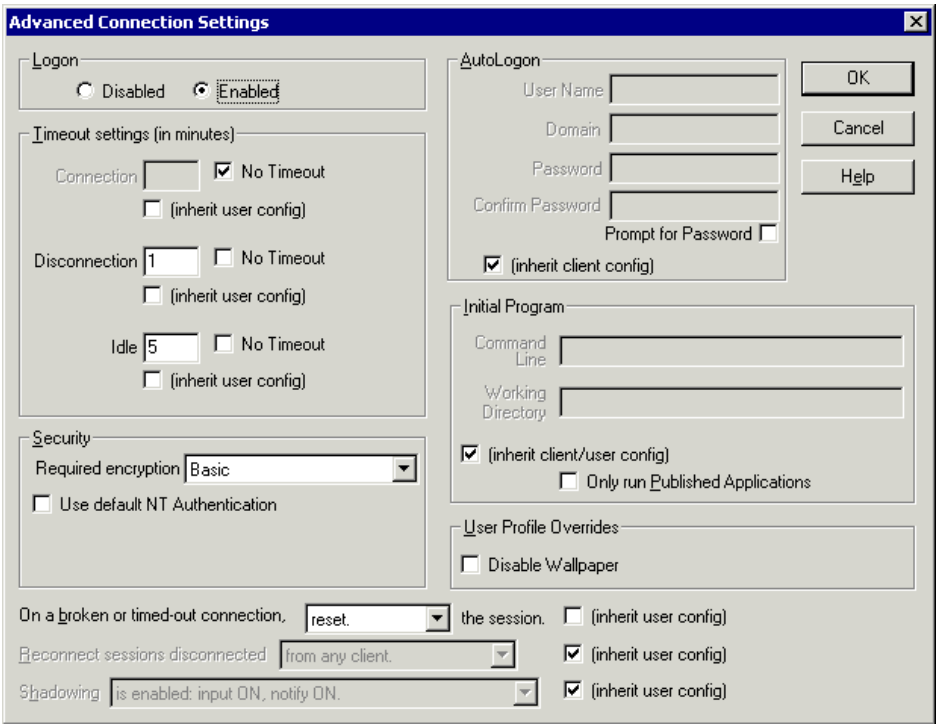
- MetaFrame 1.8 Server
- MetaFrame 3 Server

### MetaFrame 1.8 Server

**To reset the connection for every session on a MetaFrame Server:**

1. Open the Citrix Connection Configuration dialog box. Choose **Start** > **Programs** > **Citrix** > **MetaFrame** > **Citrix Connection Configuration**.
2. Double-click on the ica-tcp connection name. The Edit Connection dialog box opens.

**3** Click the **Advanced** button. The Advanced Connection Settings dialog box opens.




**4** In the bottom section of the dialog box, clear the **inherit user config** check box adjacent to the **On a broken or timed-out connection** list box. Change the entry for this list box to **reset**.

**5** Click **OK**.

### MetaFrame 3 Server

**To reset the connection for every session on a MetaFrame 3 server:**

**1** Open the Citrix Connection Configuration dialog box. Choose **Programs** > **Citrix** > **Administration Tools** > **Citrix Connection Configuration Tool**.

**2** Select the ica-tcp connection name and choose **Connection** > **Edit**. Alternatively, double-click on the connection. The Edit Connection dialog box opens.

3 Click the **Advanced** button. The Advanced Connection Settings dialog box opens.

4 In the bottom section of the dialog box, clear the **inherit user config** check box adjacent to the **On a broken or timed-out connection** list box. Change the entry for this list box to **reset**.

5 Click **OK**.

## Recording Tips

When recording a script, be sure to follow these guidelines in order to create an effective script.

### Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. If you plan to record a simple Citrix ICA session, use a single protocol script. When recording an NFUSE session, however, you must create a multi-protocol script for Citrix ICA and Web(HTML/HTTP), to enable the recording of both protocols. For more information, see Chapter 4, "Recording with VuGen."

### Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Vuser Script Sections" on page 54.

### Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

### Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start** > **Programs** > **Microsoft Word**, be sure to click on the word **Programs**.

### Don't Resize Windows

It is recommended that you do not move or resize windows during the recording session.

### Make Sure Resolution Settings are Consistent

To insure successful bitmap synchronization, make sure that the resolution settings match. On the recording machine, check the settings of the ICA client, the Recording Options, and the Run Time settings. On the Injector machines, check the settings of the ICA client, and make sure that they are consistent between all injector and recording machines.

### Use 1024 x 768 Resolution

Supported resolutions (window sizes) are 640 x 480, 800 x 600, and 1024 x 768. A settings of 1024 x 768 is recommended on the recording machine as it allows the Citrix window, whose default size is 800 x 600, to be displayed properly. Note that VuGen uses the Desktop's color settings.

### Add Manual Synchronization Points

While waiting for an event during recording, such as HTML page loading, it is recommended that you add manual synchronization points with the **ctrx_sync_on_bitmap** function. For details, see "Synchronizing Replay" on page 322.

### Disable Client Updates

Disable client updates when prompted by the Citrix client. This will prevent forward compatibility issues between VuGen and newer Citrix clients that were not yet tested.

### Windows Style

Record all windows in the "classic" windows style—not the XP style. This is relevant for **sync_on_bitmap** functions.

**To change the Windows style to "classic":**

1. Click in the desktop area.
2. Choose **Properties** from the right-click menu.
3. Select the Theme tab.

4 Choose **Windows Classic** from the Theme drop down list.

5 Click **OK**.

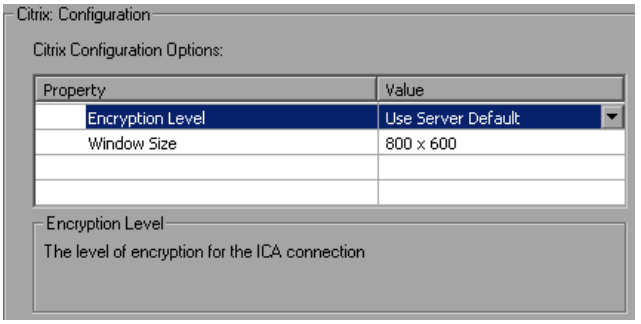
# Understanding Citrix Recording Options

You can set the Citrix Recording options in the following areas.

- Configuration
- Recorder
- Login (only for single protocol Citrix ICA scripts)

## Configuration

In the **Citrix:Configuration** Recording options, you set the window properties and encryption settings for the Citrix client during the recording session.




- **Encryption Level:** The level of encryption for the ICA connection: **Basic**, **128 bit for login only**, **40 bit**, **56 bit**, **128 bit**, or **Use Server Default** to use the machine's default.
- **Window Size:** The size of the client window: **640 x 480**, **800 x 600** (default), **1024 x 768**, **1280 x 1024**, or **1600 x 1200**.

## Recorder

The **Citrix:Recorder** Recording options let you specify how to generate window names where the window titles change during recording. You can also specify whether to save snapshots of the screens together with the script files and whether to generate text synchronization functions.

### Window Names

In some Citrix sessions, the active window name changes while you are recording. If you try to replay the script as is, the Vuser uses the original window name and the replay may fail. Using the recording options, you can specify a naming convention for the windows in which VuGen uses a common prefix or common suffix to identify the window.

For example, if the original window's name is "untitled - Notepad" where the name changes during application's run to "my_test - Notepad", you can instruct VuGen to use the common suffix only, "Notepad".

The following options are available for generating window names during recording.

**Use new window name as is**: Set the window name as it appears in the window title. (default)

**Use common prefix for new window names**: Use the common string from the beginning of the window titles, as a window name.

**Use common suffix for new window names**: Use the common string from the end of the window titles, as a name.

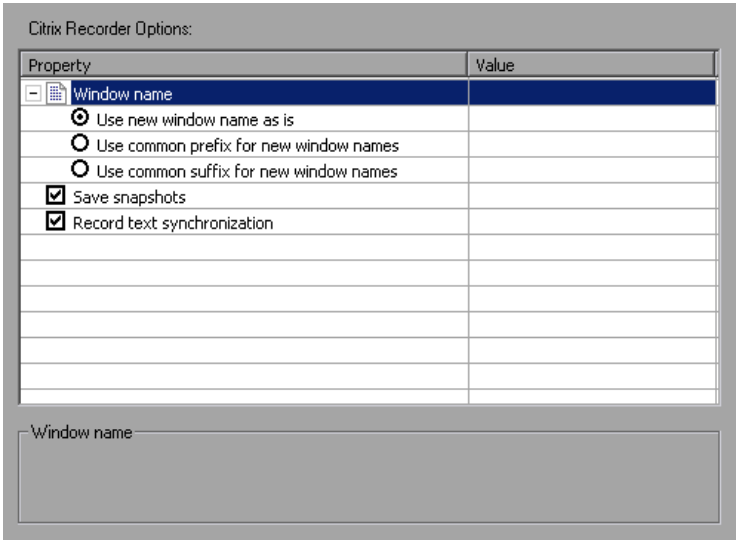
---

**Note:** Alternatively, you can modify the window names in the actual script after recording. In the Script view, locate the window name, and replace the end of the window name with the wildcard notation, *.
ctrx_sync_on_window ("My Application*", ACTIVATE, …);

---

**Save snapshots:** This option instructs VuGen to save a snapshot of the Citrix client window for each script step, when relevant. It is recommended that you enable this option to provide you with a better understanding of the recorded actions. Saving snapshots, however, uses more disk space and slows down the recording session.

**Record text synchronization:** This setting instructs VuGen to record text synchronization before each mouse click (enabled by default).




## Login

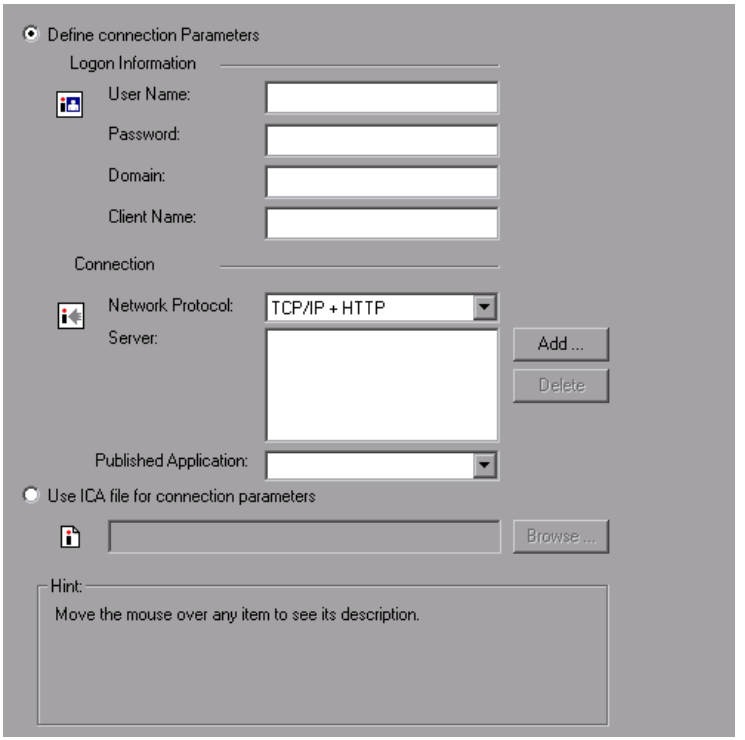
In the **Citrix:Login** Recording options, you set the connection and login information for the recording session. (When working with NFUSE, the Login options are not available since the login is done through the Web pages.)

You can provide direct login information or instruct VuGen to use an existing configuration stored in an **ica** file.

You must provide the name of the server—otherwise the connection VuGen generates a **ctrx_connect_server** function:

```
ctrx_connect_server("steel", "test", "test", "testlab");
```

If you do not provide login information, you are prompted for the information when the client locates the specified server.




### Defining Connection Parameters

**Connection** section—enter the connection— information:

- the **User Name** for the Citrix server
- the **Password** for the Citrix server
- the **Domain** of the Citrix server
- the **Client Name**, by which the MetaFrame server identifies the client (optional).

**Logon Information** section—enter the login information:

- the preferred **Network Protocol**: TCP/IP or TCP/IP+HTTP. If you intend to use a browser, choose the TCP/IP+HTTP option. For all other applications, choose TCP/IP.

- the Citrix **Server** name. To add a new server to the list, click **Add**, and enter the server name (and its port for TCP/IP + HTTP). Note that multiple servers apply only when you specify a Published Application. If you are connecting to the desktop without a specific application, then list only one server.
- the name of the **Published Application** as it is recognized on Citrix server. The drop-down menu contains a list of the available applications. If you do not specify a published application, VuGen uses the server's desktop.

  Note that if you do not specify a published application, Citrix load balancing will not work. To use load balancing when accessing the server's desktop, register the desktop as a published application on the server machine, and select this name from the **Published Application** drop-down list.

### Use ICA File for Connection Parameters

If you have an existing .ica file with all of the relevant configuration information, select **Use ICA file for connection parameters**. In the following row, specify the full path of the .ica file.

For information about the format of an ICA file, see "Understanding ICA Files" on page 328.

## Setting the Citrix Recording Options

Before recording, you set the desired recording options.

**To set the Citrix recording options:**

1. Open the Recording Options dialog box. Choose **Tools** > **Recording Options** or click the **Options** button in the Start Recording dialog box. The keyboard shortcut key is CTRL+F7.
2. Select the **Citrix:Login** node. (only for single protocol Citrix ICA scripts)
   - ➤ If you have an existing ica file with all of the relevant configuration information, select **Use ICA file for connection parameters**. Specify the full path of the ica file, or click the **Browse** button and locate the file on the local disk or network.
   - ➤ If you do not have an ica file, select **Define connection parameters**. This is the default setting. Enter the **Connection** and **Identification** information.
3. Select the **Citrix:Configuration** node. Choose an encryption level and a window size.
4. Select the **Citrix:Recorder** node. Specify how to generate window names for windows whose titles change during the recording session.
5. To prevent VuGen from saving a snapshot for each step, clear **Save snapshots**.
6. When recording an NFUSE session, set the Web recording mode to URL-based. Choose the **Internet Protocol:Recording** recording option and select **URL-based script**.
7. Click **OK** to accept the setting and close the dialog box.

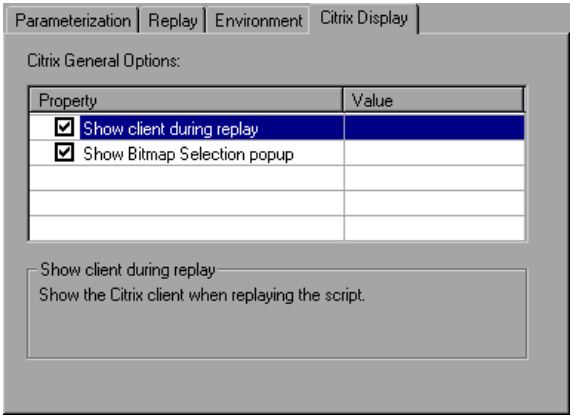
# Setting the Citrix Display Settings

Before running your Citrix Vuser script, you can set several display options to be used during replay. Although these options increase the load upon the server, they are useful for debugging and analyzing your session.

**To set the Citrix display options:**

1 Open the General Options dialog box. Choose **Tools** > **General Options** in the main VuGen window.

2 Select the **Citrix Display** tab.




3 Select **Show client during replay** to display the Citrix client when replaying the Vuser script.

4 Select **Show Bitmap Selection popup** to issue a popup message when you begin to work interactively within a snapshot. VuGen issues this message when you choose the right-click menu option **Insert Sync Bitmap** or **Insert Get Text**, before you select the bitmap or text.

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 on 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 IP spoofing.

---

To set the General Run-time settings, see Chapter 12, "Configuring Run-Time Settings." To set the Speed Emulation properties, see Chapter 13, "Configuring Network Run-Time Settings."

You can set the Citrix-specific run-time settings in the following areas:

- Citrix Configuration Run-Time Settings
- Citrix Timing Run-Time Settings

## Citrix Configuration Run-Time Settings

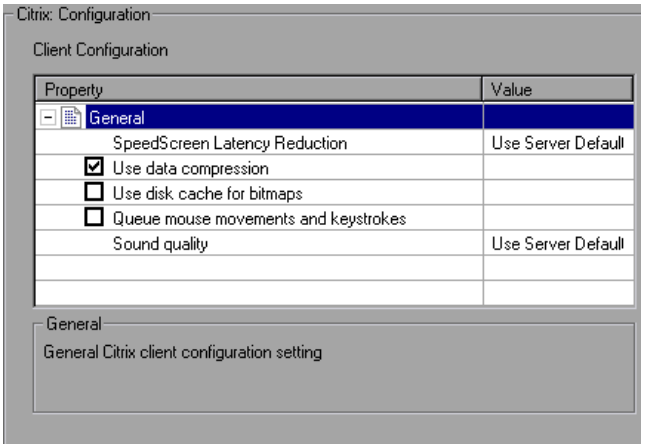
The configuration settings relate to the screen latency, data compression, disk cache, and queuing of mouse movements.

**To set the Configuration Run-Time Settings:**



**1** Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the VuGen toolbar, or 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. This serves to reduce 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 Select one of the options for **Sound quality** from the list: **Use server default**, **Sound off**, **High sound quality**, **Medium sound quality**, or **Low sound quality**. If the client machine does not have a 16-bit Sound Blaster-compatible sound card, select **Sound Off**. With sound support enabled, you will be able to play sound files from published applications on your client machine.

### Citrix Timing Run-Time Settings

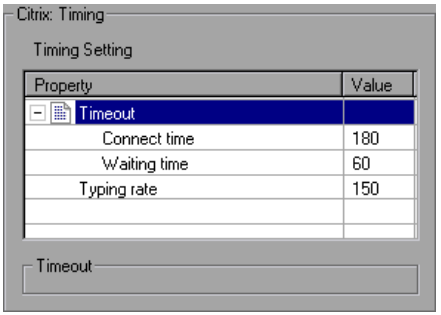
The timeout settings relate to the connect and waiting times.

**To set the Timing 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:Timing** node.




3 Indicate the **Connect Time**, the time in seconds to wait idly at an established connection before exiting. The default is 180 seconds.

4 Indicate the **Waiting Time**, the time in seconds to wait idly at a synchronization point before exiting. The default is 60 seconds.

5 Specify a **Typing rate**, the delay in milliseconds between keystrokes.

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

# Viewing and Modifying Citrix Vuser Scripts

You can view the contents of your Vuser script in VuGen's Script view or Tree view. For general information about viewing a script, see Chapter 2, "Introducing VuGen."

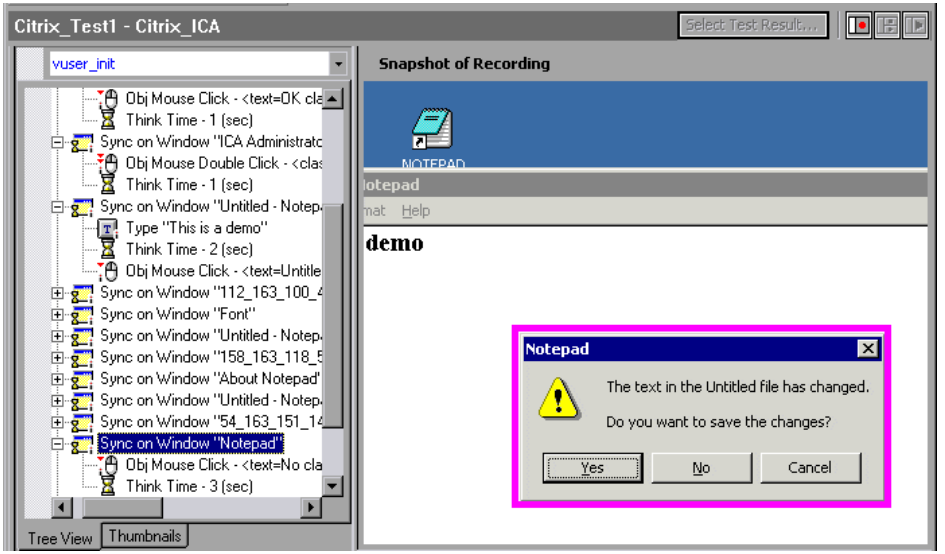
In Tree view, you can view a Citrix Vuser's snapshots. The following Citrix steps have snapshots associated with them:

- **Obj Mouse Click**, **Obj Mouse Double Click**, and **Obj Mouse Down**
- **Sync on Window** and **Sync on Bitmap**

In addition to displaying the client window, the snapshot also highlights the object upon which the action was performed.

- For the **Mouse** steps, a small pink square indicates where the user clicked.
- For **Sync on Bitmap**, a pink box encloses the bitmap area.

For **Sync on Window**, a pink box encloses the entire window. In the following example, the snapshot shows the **Sync On Window** step. Notepad's confirmation box is enclosed by a box indicating the exact window on which the operation was performed.



Note that VuGen saves snapshots as bitmap files in the script's **data\snapshots** directory. You can determine the name of the snapshot file by checking the function's arguments.

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

After recording, you can manually add steps to the script in either Script view and Tree View. For information about the various script views, see "Viewing and Modifying Vuser Scripts" on page 17.

In addition to manually adding new functions, you can add new steps interactively for Citrix Vusers, directly from the snapshot. Using the right-click menu, you can add bitmap and text-related steps. Several additional steps are also available from the right-click menu when the **agent** is installed. For more information, see Chapter 22, "Using the LoadRunner Citrix Agent."

**To insert a function interactively:**

1. Click on a step within Tree view. Make sure that a snapshot is visible.
2. Click within the snapshot.
3. Right-click and choose one of the commands. A dialog box opens with the step's properties.
4. Modify the desired properties and click **OK**. VuGen inserts the step into your script.

## Synchronizing Replay

When running a script, it is often necessary to synchronize the actions to insure a successful replay. Synchronization refers to the timing of events within your script, waiting for windows and objects to become available before executing an action. For example, you may want to check whether a certain window has opened before attempting to press a button.

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: **ctrx_sync_on_window** and with an agent installation (see Chapter 21, "Creating Citrix Vuser Scripts") **ctrx_sync_on_text**.

### Sync on Window

The **ctrx_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 **ctrx_sync_on_window** function, you can view the snapshot from the script's Tree view.

### Sync on Text

**ctrx_sync_on_text** is a synchronization function that waits for a specified string to appear at the specified position before continuing. The function searches for the text in a radius of forty pixels in all directions around the specified coordinates.

VuGen records **ctrx_sync_on_text** before every mouse click or double-click, when the LoadRunner Citrix Agent is installed.

The following segment shows a **ctrx_sync_on_text** function that was recorded during a Citrix recording with the LoadRunner Citrix Agent installed.

```
ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0,
0,391,224, "snapshot1", CTRX_LAST);

ctrx_sync_on_text (196, 198, "OK", TEXT, "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);

ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198,
LEFT_BUTTON, 0, "ICA Seamless Host Agent=snapshot2",
CTRX_LAST);
```

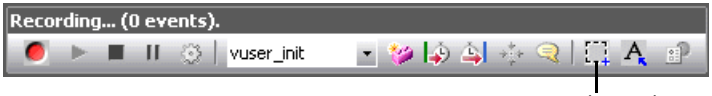
For more information on this function, refer to the *Online Function Reference* (**Help** > **Function Reference**).

## Manual Synchronization

You can also add manual synchronization during recording either through VuGen's user interface or by inserting custom synchronization functions after recording. A common use of this capability is where the actual window did not change, but an object within the window did change. Since the window did not change, VuGen did not detect or record a **Sync on Window.** For example, if you want the replay to wait for a specific graphic image in a browser window, you insert manual synchronization. Or, if you are recording a large window with several tabs, you can insert a synchronization step to wait for the new tab's content to open.

### Synchronizing During Recording

To add synchronization during recording, you use the floating toolbar's marker tool. The marker tool lets you to mark an area within the client window that needs to be in focus before resuming replay.




*Marker Tool*

**To mark a bitmap area for synchronization:**



1 Click the **marker** button.

2 Drag the mouse from the top left of the section of the area to the bottom right. In Tree view, VuGen generates a **Sync on Bitmap** step after the current step. In Script view, VuGen generates a **ctrx_sync_on_bitmap** function with the selected coordinates as arguments.

```
ctrx_sync_on_bitmap(93, 227, 78, 52,
                    “66de3122a58baade89e63698d1c0d5dfa”);
```

During replay, Vusers look for the bitmap at the specified coordinates, and wait until it is available before resuming the test.

### Synchronizing After Recording

You can also add synchronization after the recording session. To implement additional types of synchronization, you must manually enter one of the synchronization steps into your script. To insert a function, choose Insert > Add Step and choose the desired function. For more information, see Chapter 7, “Enhancing Vuser Scripts.”

### Sync on Bitmap

- **Sync on Obj Info** (agent installations only)
- **Sync on Text** (agent installations only)

During recording, the bitmaps generated for the **Sync on Bitmap** step are saved under the script’s **data\snapshots** directory. If synchronization fails during replay, VuGen generates a new bitmap that you can examine to determine why synchronization failed. The bitmap name has the format of sync_bitmap_<hash_value>.bmp. It is stored in the script’s output directory, or for a scenario, profile, or tuning session, wherever the output files are written.

In addition, you can add several other steps that affect the synchronization indirectly:

- **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 **ctrx_sync_on_window** functions are timing out, you can increase the default timeout of 60 seconds to 180.

- **Win Exist**: Checks if a window is visible in the Citrix client. By adding control flow statements, you can use this function to check for a window that does not always open, such as a warning dialog box. In the following example, **ctrx_win_exist** checks whether a browser was launched. The second argument indicates how long to wait for the browser window to open. If it did not open in the specified time, it double-clicks its icon.

```
if (!ctrx_win_exist("Welcome to MSN.com- Microsoft Internet Explorer",6))
  ctrx_mouse_double_click(34, 325, LEFT_BUTTON, 0)
```

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

### Waiting for a Bitmap Change

In certain cases, you do not know what data or image will be displayed in an area, but you do expect it to change. To emulate this, you can use the **ctrx_sync_on_bitmap_change** function. You specify an area (coordinates and size), in which VuGen waits for a change. To assist you in deriving the correct coordinates for the bitmap area, you can record a **Sync on Bitmap** step using the marker tool (see above), and manually modify the function name and remove the fifth argument.

The syntax of the functions is 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,]
[initial_bitmap_value,] CTRX_LAST);

You can add optional arguments to **ctrx_sync_on_bitmap_change**:

- initial wait time value—when to begin checking for a change.
- a timeout—the amount of time in seconds to wait for a change to occur before failing.

➤ initial bitmap value—the initial hash value of the bitmap. Vusers wait until the hash value is different from the specified initial bitmap value.

```
/* recorded function */
ctrx_sync_on_bitmap(93, 227, 78, 52,
                    “66de3122a58baade89e63698d1c0d5dfa”);
```

```
/* modified function with an initial wait time of 300 and timeout of 400*/
ctrx_sync_on_bitmap_change(93, 227, 78, 52, 300, 400, CTRX_LAST);
```

---

**Note:** If you are using **Sync on Bitmap**, make sure that the Configuration settings in the Controller/Console, Load Generator machine, and screen are the same. Otherwise, VuGen may be unable to find the correct bitmaps during replay. For information on how to configure the client settings, see “Configuration” on page 311.

---

## 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
```

## Using Citrix Functions

During a Citrix recording session, VuGen generates functions that emulate the communication between a client and a remote server. The generated functions have a **ctrx** prefix. You can also manually edit any of the functions into your Vuser script after the recording session. For more information about the **ctrx** 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.

### Connection Functions

| | |
|---|---|
| **ctrx_connect_server** | Connects to a remote server using the Citrix client. |
| **ctrx_disconnect_server** | Closes the connection to a server. |
| **ctrx_nfuse_connect** | Connects to a Citrix server via an NFUSE portal. |
| **ctrx_set_connect_opt** | Sets the connection options. |

### Mouse Functions

| | |
|---|---|
| **ctrx_mouse_click** | Emulates a mouse click. |
| **ctrx_mouse_double_click** | Emulates a mouse double-click. |
| **ctrx_mouse_down** | Emulates the pressing of a mouse button. |
| **ctrx_mouse_up** | Emulates the release of a mouse button. |

### Object Functions (Agent Only)

| | |
|---|---|
| **ctrx_obj_get_info** | Gets class information about an object. |
| **ctrx_obj_mouse_click** | Emulates a mouse click for the specified object. |
| **ctrx_obj_mouse_double_click** | Emulates a mouse double-click for the specified object. |
| **ctrx_obj_mouse_down** | Emulates the pressing of a mouse button for the specified object. |
| **ctrx_obj_mouse_up** | Emulates the releasing of a mouse button for the specified object. |
| **ctrx_sync_on_obj_info** | Waits for the specified object to be created or active. |

### Synchronization Functions

| | |
|---|---|
| **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_obj_info** (with agent only) | Waits for the specified object to be created or active. |
| **ctrx_sync_on_text** (with agent only) | Waits until certain text is displayed at the specified position. |
| **ctrx_sync_on_window** | Waits for a window to be created or active. |
| **ctrx_unset_window** | Waits for the specified window to close. |
| **ctrx_wait_for_event** | Waits for the specified event to occur. |
| **ctrx_win_exist** | Checks whether the specified window exists. |

### Keyboard Functions

| | |
|---|---|
| **ctrx_key** | Emulates the typing of a non-alphanumeric key. |
| **ctrx_key_down** | Emulates the pressing of a key on the keyboard. |
| **ctrx_key_up** | Emulates the releasing of a keyboard key. |
| **ctrx_type** | Emulates the typing of an alphanumeric key. |

### Information Retrieval Functions

| | |
|---|---|
| **ctrx_button_get_info** | Retrieves class information about a button. |
| **ctrx_edit_get_info** | Retrieves class information about an edit field. |
| **ctrx_get_bitmap_value** | Gets the hash value of the specified bitmap. |
| **ctrx_get_text** (with agent only) | Stores the demarcated text in a buffer. |
| **ctrx_get_window_name** | Gets the name of the window in focus. |
| **ctrx_get_window_position** | Gets the position of the specified window, or of the window in focus. |
| **ctrx_list_get_info** (with agent only) | Retrieves class information about a list. |
| **ctrx_list_select_item** (with agent only) | Selects an item from a list. |
| **ctrx_menu_select_item** (with agent only) | Selects a menu item. |
| **ctrx_obj_get_info** (with agent only) | Gets class information about an object. |

### Selection Functions

| | |
|---|---|
| **ctrx_list_select_item** | Selects an item from a list. |
| **ctrx_menu_select_item** | Selects a menu item. |

### General Functions

| | |
|---|---|
| **ctrx_save_bitmap** | Saves the demarcated bitmap in a buffer. |
| **ctrx_set_exception** | Specifies exception handling. |

# Tips for Replaying and Troubleshooting Citrix Vuser Scripts

The following sections provide guidelines and tips for Citrix Vusers in the following areas:

- Replay Tips
- Debugging Tips

For recording tips, see "Recording Tips" on page 309.

## Replay Tips

### Set Initialization Quota

To prevent overloading by multiple Vusers while connecting, set an initialization quota of 4 to 10 Vusers (depending on the capacity of the server) or apply ramp-up initialization using the Scheduler.

### Enable Think Time

For best results, do not disable think time in the Run-Time settings. Think time is especially relevant before the **ctrx_sync_on_window** and **ctrx_sync_on_bitmap** functions, which require time to stabilize.

### Set Consistency Between Machines

If you intend to replay the script on another machine, make sure that the following items are consistent between the record and replay machines: Window Size (resolution), Window Colors, System Font and the other Default Options settings for the Citrix client. These settings affect the hash value of bitmaps, and inconsistencies may cause replay to fail. To view the Citrix Client settings, select an item from the Citrix program group and choose **Application Set Settings** or **Custom Connection Settings** from the right-click menu. Select the Default Options tab.

### Increasing the Numbers of Vusers per Load Generator Machine

Load Generator machines running Citrix Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine, also known as the GDI (Graphics Device Interface). To increase the number of Vusers per machine, you can open a terminal server session on the machine which acts as an additional injector machine.

The GDI count is Operating System dependent. The actual GDI (Graphics Device Interface) count for a heavily loaded machine using LoadRunner is approximately 7,500. The maximum available GDI on Windows 2000 machines is 16,384.

For more information on creating a terminal server session, see the Terminal Services topics in the *LoadRunner Controller User's Guide*.

---

**Note:** By default, sessions on a terminal server use a 256-color set. If you intend to use a terminal session for load testing, make sure to record on machines with a 256-color set.

---

## Debugging Tips

### Single Client Installation

If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine. To verify that only one client is installed, open the Add/Remove Programs dialog box from the Control Panel and make sure that there is only one entry for the Citrix ICA client.

### Add Breakpoints

Add breakpoints to your script in VuGen to help you determine the problematic lines of code.

### Synchronize Your Script

If replay fails, you may need to insert synchronization functions into your script to allow more time for the desired windows to come into focus. Although you can manually add a delay using **lr_think_time**, it is recommended that you use one of the synchronization functions discussed in "Synchronizing Replay" on page 322.

### Extended Log

You can view additional replay information in the Extended log. To do this, enable Extended logging in the Run-Time settings (F4 Shortcut key) **Log** tab. You can view this information in the Execution Log tab or in the output.txt file in the script's directory.

### Snapshot Bitmap

When an error occurs, VuGen saves a snapshot of the screen to the script's **output** directory. You can view the bitmap to try to determine why the error occurred.

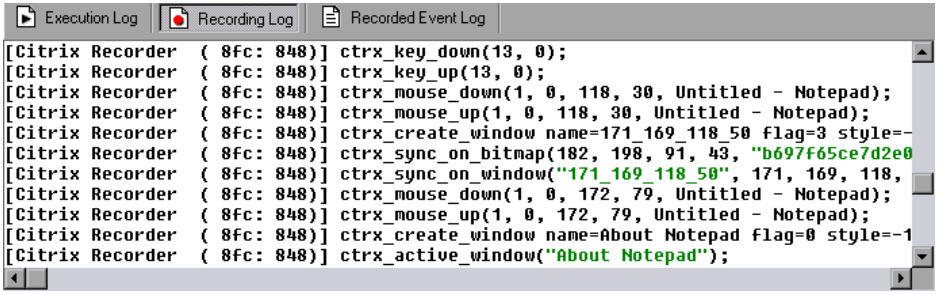
During recording, the bitmaps generated for the **ctrx_sync_on_bitmap** function are saved under the script's **data** directory. The bitmap name has the format of hash_value.bmp. If synchronization fails during replay, the generated bitmap is written to the script's output directory, or if you are running it in a scenario or tuning session, to wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.

### Show Vusers

To show Vusers during a scenario or tuning session run, enter the following in the Vuser command line box: -lr_citrix_vuser_view. In the Controller or Tuning Module Console, 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.

### View Output Window

To see detailed information about the recording, view the recording log in the Output window. Choose **View** > **Output Window** and select the **Recording Log** tab. VuGen displays a detailed log of all actions performed by VuGen.



# 22

# Using the LoadRunner Citrix Agent

The LoadRunner Citrix Agent is an optional utility that you can install on the Citrix server. It provides you with several important benefits:

- Intuitive and readable scripts
- Built-in synchronization
- Detailed Information about all objects
- Ability to work interactively within the Client window

This chapter contains the following sections:

- About the LoadRunner Citrix Agent
- Benefitting From the Citrix Agent
- Installation
- Effects and Memory Requirements of the Citrix Agent
- Sample Script

***The following information only applies to the Citrix ICA protocol.***

## About the LoadRunner Citrix Agent

The LoadRunner Citrix Agent is an optional utility that you can install on the Citrix server. It is provided on the LoadRunner CD and you can install it on any Citrix server.

The agent provides Load Generator machines with detailed information about objects and events in the client window. It also lets you work interactively within the client screen to add object-specific steps.

## Benefitting From the Citrix Agent

The Citrix agent provides enhancements in the following areas:

- Object Details: provides detailed information about individual objects in the Citrix client window.
- Active Object Recognition: shows you which objects in the client window are recognized by VuGen.
- Expanded Right-Click Menu: additional right-click menu items that allow you to add synchronization, verification, and text retrieval steps.
- Retrieving Text: capability to insert text searches to your script.

### Object Details

When the Citrix agent is installed, VuGen records specific information about the active object instead of a general information about the action. For example, VuGen generates **Obj Mouse Click** and **Obj Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows the same mouse-click action recorded with and without the agent installation. Note that with an agent, VuGen generates **ctrx_obj_xxx** functions for all of the mouse actions, such as click, double-click and release

```
/* Without Agent Installation */
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0, test3.txt - Notepad);

/* With Agent Installation */
ctrx_obj_mouse_click("<text=test3.txt - Notepad class=Notepad>" 573,
      61, LEFT_BUTTON, 0, test3.txt - Notepad=snapshot21,
CTRX_LAST);
```

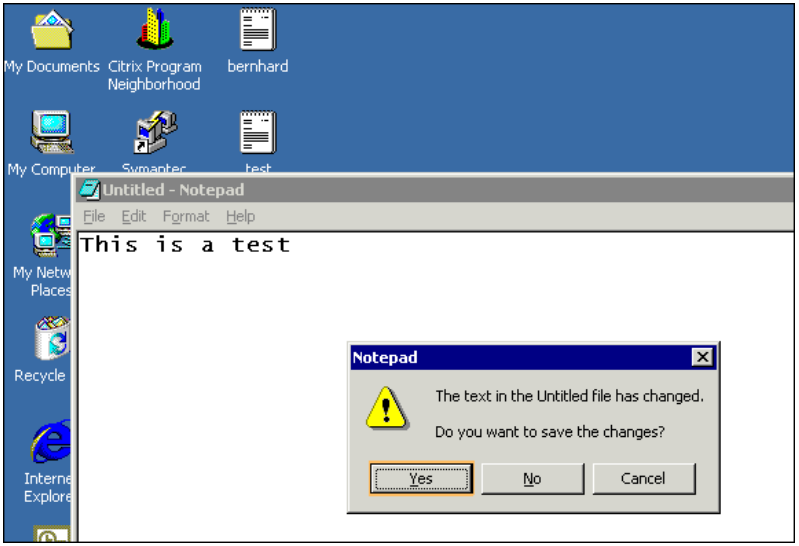
In the example above, the first argument of the **ctrx_obj_mouse_click** function contains the text of the window's title and the class, Notepad. Note that although the agent provides additional information about each object, Vusers only access objects by their window name and its coordinates.

## Active Object Recognition

The agent installation lets you see which objects in the client window are detected by VuGen. This includes all Windows Basic Objects such as edit boxes, buttons, and item lists in the current window.

To see which objects were detected, you move your mouse through the snapshot. VuGen highlights the borders of the detected objects as the mouse passes over them.

In the following example, the **Yes** button is one of the detected objects.



### Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When no agent is installed, you are limited to the **Insert Mouse Click**, **Insert Mouse Double Click**, and **Insert Sync on Bitmap**. Note that the option to add an **Insert Sync on Bitmap** step from the right-click menu is not available if you are using a 256-color set.

When an agent is installed, the following additional options are available within the window in focus, from the right-click menu: **Insert Get Text, Insert Obj Get Info**, and **Insert Sync on Obj Info**. These commands are interactive—when you insert them into the script, you mark the object or text area in the snapshot.

The **Insert Obj Get Info** and **Insert Sync on Obj Info** steps provide information about the state of the object: ENABLED, FOCUSED, VISIBLE, TEXT, CHECKED, and LINES. The **Insert Sync on Obj Info** step, generated as a **ctrx_sync_on_obj_info** function, instructs VuGen to wait for a certain state before continuing. The **Insert Obj Get Info** step, generated as a **ctrx_get_obj_info** function, retrieves the current state of any object property. The **Insert Get Text** step is discussed in the section "Retrieving Text" on page 341.

In the following example, the **ctrx_sync_on_obj_info** function provides synchronization by waiting for the Font dialog box to come into focus.

```
ctrx_sync_on_obj_info("Font", 31, 59, FOCUSED, "TRUE", CTRX_LAST);
```

Utilizing VuGen's ability to detect objects, you can perform actions on specific objects interactively, from within the snapshot.

**To insert a function interactively using the agent capabilities:**

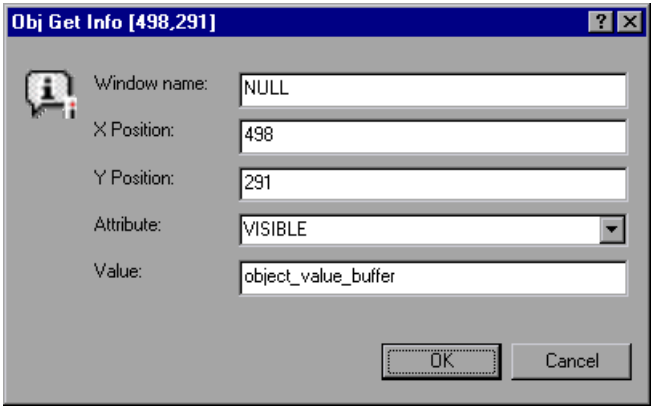
1. Click at a point within the tree view to insert the new step. Make sure that a snapshot is visible.
2. Click within the snapshot.
3. To mark a bitmap, right-click on it and choose **Insert Sync on Bitmap**.

   VuGen issues a message indicating that you need to mark the desired area by dragging the cursor. Click **OK** and drag the cursor diagonally across the bitmap that you want to select.

When you release the mouse, VuGen inserts the step into the script after the currently selected step.

**4** For all other steps, move your mouse over snapshot objects to determine which items are active—VuGen highlights the borders of active objects as the mouse passes over them.

Right-click and choose one of the Insert commands. A dialog box opens with the step's properties.




Set the desired properties and click **OK**. VuGen inserts the step into your script.

### Retrieving Text

With the agent installed, VuGen lets you save standard text to a buffer. Note that VuGen can only save true text—-not a graphical representation of text in the form of an image.

You save the text using the **Get Text** step either during or after recording. During recording, VuGen displays an additional **Get Text** toolbar button.




After recording, you insert the **Get Text** step from the snapshot's right-click menu.

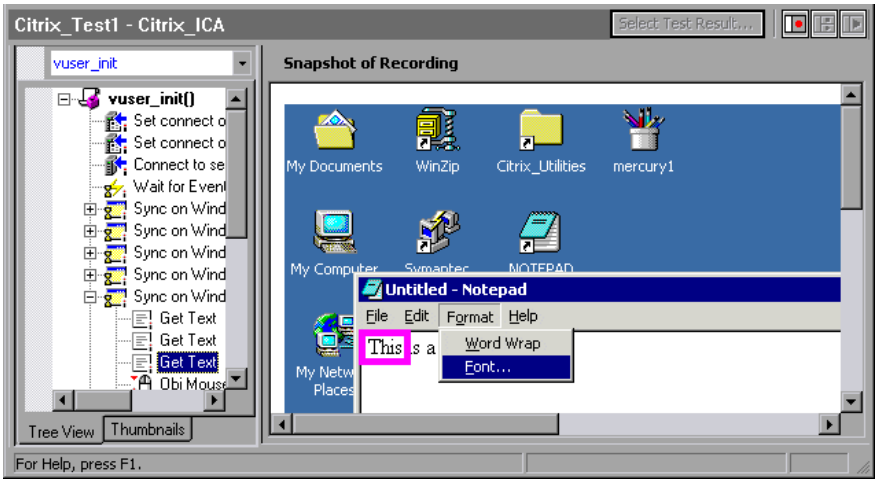
**To retrieve a text string:**

1 During recording: Click the **Get Text** button.

After recording: Choose **Insert Get Text** from the snapshot's right-click menu. The Bitmap Selection dialog opens, indicating that you are inserting a synchronization or informational function and that you need to mark an area.

2 Click at the corner of the text that you want to capture, drag the mouse diagonally to mark the text you want to save, and release the mouse button.

VuGen places a **Get Text** step at the current location and saves the text to a buffer. VuGen marks the saved text with a pink box. In the following snapshot, the **Get Text** step retrieved the text **This**.



# Installation

The installation file for the Citrix Agent is located on the LoadRunner CD #2, under the Additional Components\CitrixAgent folder. The disk space required for installing the Citrix agent is 25 MBs.

Note that the agent should only be installed on your Citrix server machine—not Load Generator machines.

If you are upgrading a Citrix Agent, make sure to uninstall the previous version before installing the next one (see uninstallation instructions below).

**To install the Citrix Agent:**

1. If your server requires administrator permissions to install software, log in as an administrator to the server.
2. Locate the installation file, **CitrixAgent.exe**, on the LoadRunner CD #2 in the Additional Components\CitrixAgent folder.

---

**Note:** After installation the agent will only be active for LoadRunner invoked Citrix sessions—it will not be active for users who start a Citrix session without LoadRunner.

---

To disable the Citrix Agent, you must uninstall it.

**To uninstall the Citrix Agent:**

1. If your server requires administrator privileges to remove software, log in as an administrator to the server.
2. Choose **Start** > **LoadRunner Citrix Agent** > **Uninstall LoadRunner Citrix Agent** and follow the uninstall instructions.

   Alternatively, open **Add/Remove Programs** in the server machine's Control Panel. Select **LoadRunner Citrix Agent** and click **Change/Remove**.

## Effects and Memory Requirements of the Citrix Agent

When you run Citrix Vusers with the agent installed, each Vuser runs its own process of **ctrxagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine (about 7%).

The memory requirements per Citrix ICA Vuser (each Vuser runs its own **ctrxagent.exe** process) is approximately 4.35 MB. To run 25 Vusers, you would need 110 MBs of memory.

## Sample Script

The following script illustrates a true Citrix ICA session with an agent.

```
vuser_init ()
{
  ctrx_set_connect_opt (NETWORK_PROTOCOL, "TCP/IP + HTTP");
  ctrx_connect_server ("Plato", "test", lr_decrypt("428c4445a14409b9"),
"QAlab");
  ctrx_wait_for_event ("LOGON");
  ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0,
0,391,224, "snapshot1", CTRX_LAST);
  ctrx_sync_on_text (196, 198, "OK", TEXT, "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);
  ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198,
LEFT_BUTTON, 0, "ICA Seamless Host Agent=snapshot2",
CTRX_LAST);
  lr_think_time (73);
return 0;
}
```

# Part V

## Client Server Protocols

# 23

# 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:

- About Developing Database Vuser Scripts
- Introducing Database Vusers
- Understanding Database Vuser Technology
- Getting Started with Database Vuser Scripts
- Setting Database Recording Options
- Database Advanced Recording Options
- Using LRD Functions
- Understanding Database Vuser Scripts
- Evaluating Error Codes
- Handling Errors

***The following information only applies to Client Server Database (Sybase CTLib, Sybase DbLib, Informix, MS SQL Server, Oracle 2-Tier, ODBC, and DB2 CLI) and ERP/CRM Siebel Vuser scripts.***

# About Developing Database Vuser Scripts

When you record a database application communicating with a server, VuGen generates a Database Vuser script. VuGen supports the following database types: CtLib, DbLib, Informix, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity. Each LRD function has an **lrd** prefix and represents one or more database functions. For example, the **lrd_fetch** function represents a fetch operation.

When you run a recorded session, the Vuser script communicates directly with the database server, performing the same operations as the original user. You can set the Vuser behavior (run-time settings) to indicate the number of times to repeat the operation and the interval between the repetitions. For more information, see Chapter 12, "Configuring Run-Time Settings."

Using VuGen, you can parameterize a script, replacing recorded constants with parameters. For more information, see Chapter 8, "Working with VuGen Parameters."

In addition, you can correlate queries or other database statements in a script, linking the results of one query with another. For more information, see Chapter 11, "Correlating Statements."

For troubleshooting information and scripting tips, see Chapter 78, "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.

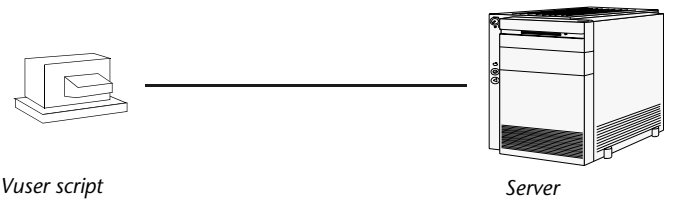
The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. The Vusers execute the script and emulate user load on the client/server system. The Vusers generate performance data which you can analyze in report and graph format.

# Understanding Database Vuser Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.




Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.




You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and UNIX environments. For information about recording scripts, see Chapter 4, “Recording with VuGen.”

Users working in a UNIX only environment can create Database Vuser scripts through programming using VuGen templates as the basis for a script. For information about programming Database Vuser scripts on UNIX, see Appendix C, “Programming Scripts on UNIX Platforms.”

# Getting Started with Database Vuser Scripts

This section provides an overview of the process of developing Database Vuser scripts using VuGen.

**To develop a Database Vuser script:**

**1 Record the basic script using VuGen.**

Invoke VuGen and create a new Vuser script. Specify the type of Vuser (**Client Server** or **ERP** protocol types). Choose an application to record and set the recording options. Record typical operations on your application.

For details, see Chapter 4, "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 7, "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 8, "Working with VuGen Parameters."

**4 Correlate queries (optional).**

Correlating database statements allows you to use the result of a query in a subsequent one. This feature is useful when working on a database with user constraints.

For details, see Chapter 11, "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 12, "Configuring Run-Time Settings."

**6 Run the script from VuGen.**

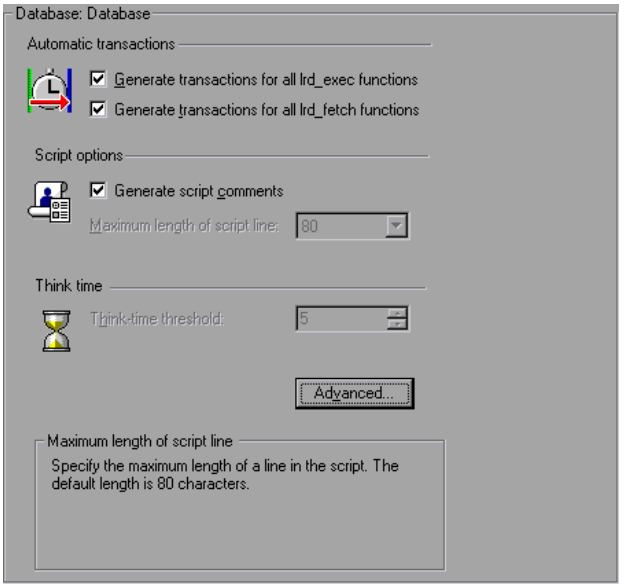
Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 14, “Running Vuser Scripts in Standalone Mode.”

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## 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.

**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.

# Database Advanced Recording Options

In addition to the basic recording options, you can configure advanced options for the log file detail, CtLib specific functions, buffer size, and the recording engine.

**Recording Log Options:** You can set the detail level for the trace and ASCII log files. The available levels for the trace file are **Off**, **Error Trace**, **Brief Trace**, or **Full Trace**. The error trace only logs error messages. The Brief Trace logs errors and lists the functions generated during recording. The Full Trace logs all messages, notifications, and warnings.

You can also instruct VuGen to generate ASCII type logs of the recording session. The available levels are **Off**, **Brief detail**, and **Full detail**. The Brief detail logs all of the functions, and the Full detail logs all of the generated functions and messages in ASCII code.

**CtLib Function Options:** You can instruct VuGen to generate a send data time stamp or an extended result set statement.

- **Time Stamp:** By default, VuGen generates **lrd_send_data** statements with the **TotalLen** and **Log** keywords for the **mpszReqSpec** parameter. The Advanced Recording Options dialog box lets you instruct VuGen to also generate the **TimeStamp** keyword. If you change this setting on an existing script, you must regenerate the Vuser script by choosing **Tools** > **Regenerate Script.** It is not recommended to generate the **Timestamp** keyword by default. The timestamp generated during recording is different than that generated during replay and script execution will fail. You should use this option only after a failed attempt in running a script, where an **lrd_result_set** following an **lrd_send_data** fails. The generated timestamp can now be correlated with a timestamp generated by an earlier **lrd_send_data**.

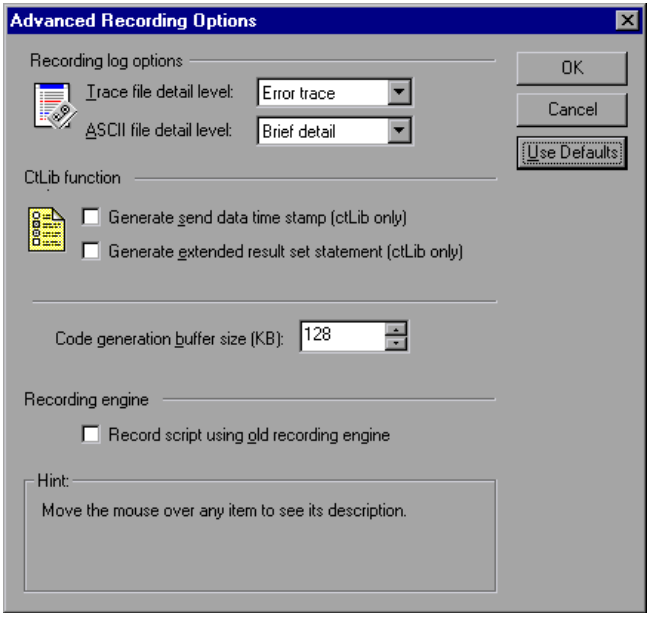
➤ **Extended Result Set:** By default, VuGen generates an **lrd_result_set** function when preparing the result set. This setting instructs VuGen to generate the extended form of the **lrd_result_set** function, **lrd_result_set_ext**. In addition to preparing a result set, this function also issues a return code and type from **ct_results**.

**Code Generation Buffer Size:** Specify in kilobytes the maximum size of the code generation buffer. The default value is 128 kilobytes. For long database sessions, you can specify a larger size.

**Recording Engine:** You can instruct VuGen to record scripts with the older LRD recording engine for compatibility with previous versions of VuGen. This option is only available for single-protocol scripts.

**To set advanced recording options:**

1 Click the **Advanced** button in the Database node of the Recording Options dialog box. The Advanced Recording Options dialog box opens.




2 Select a **Trace file detail level**. To disable the trace file, select **Off**.

3 To generate an ASCII log file, select the desired detail level from the **ASCII file detail level** box.

4 For CtLib: To instruct VuGen to generate the TimeStamp keyword for **lrd_send_data** functions, select the **Generate send data time stamp** option.

5 For CtLib: To instruct VuGen to generate **lrd_result_set_ext** instead of **lrd_result_set**, select the **Generate extended result set statement** option.

6 To modify the size of the code generation buffer from the default value of 128 kilobytes, enter the desired value in the **Code generation buffer size** box.

7 To use the old VuGen recording engine to allow backwards compatibility, select the **Record script using old recording engine** option.

8 Click **OK** to save your settings and close the Advanced Recording Options dialog box.

## Using LRD Functions

The functions developed to emulate communication between a database client and a server are called LRD Vuser functions. Each LRD Vuser function has an **lrd** prefix. VuGen automatically records most of the LRD functions listed in this section during a database session (CtLib, DbLib, Informix, Oracle (2-Tier), and ODBC). You can also manually program any of the functions into your script. For syntax and examples of the LRD functions, refer to the *Online Function Reference* (**Help > Function Reference**).

### Access Management Functions

| | |
|---|---|
| **lrd_alloc_connection** | Allocates a connection structure. |
| **lrd_close_all_cursors** | Closes all open cursors. |
| **lrd_close_connection** | Disconnects (logs out) from the database. |
| **lrd_close_context** | Closes a context. |
| **lrd_close_cursor** | Closes a database cursor. |
| **lrd_ctlib_cursor** | Specifies a CtLib cursor command. |
| **lrd_commit** | Commits the current transaction. |
| **lrd_db_option** | Sets an option for the current database. |
| **lrd_free_connection** | Frees a connection structure. |

| | |
|---|---|
| **lrd_rollback** | Rolls back the current transaction. |
| **lrd_open_connection** | Connects (logs on) to the database. |
| **lrd_open_context** | Opens a context. |
| **lrd_open_cursor** | Opens a database cursor. |

### LRD Environment Functions

| | |
|---|---|
| **lrd_msg** | Issues an output message. |
| **lrd_option** | Sets an LRD option. |
| **lrd_end** | Closes the lrd environment. |
| **lrd_init** | Initializes the lrd environment. |

### Retrieval Handling Functions

| | |
|---|---|
| **lrd_col_data** | Sets a pointer indicating the location of data. |
| **lrd_fetch** | Fetches the next row in the result set. |
| **lrd_fetchx** | Fetches the next row in the result set using an extended fetch. (ODBC only) |
| **lrd_result_set** | Returns a result set. (CtLib only) |
| **lrd_result_set_ext** | Returns a CtLib result code and result type (extended). |
| **lrd_fetch_adv** | Fetches multiple rows from a result set using an extended fetch. (ODBC only) |
| **lrd_reset_rows** | Prepares fetched rows for an Update operation. (ODBC only) |
| **lrd_row_count** | Returns the number of the rows affected by an UPDATE, DELETE or INSERT statement. (ODBC, DB2) |

### Statement Handling Functions

| | |
|---|---|
| **lrd_bind_col** | Binds a host variable to an output column. |
| **lrd_bind_cols** | Binds a host variable array to columns. |
| **lrd_bind_cursor** | Binds a cursor to a place holder. |
| **lrd_bind_placeholder** | Binds a host variable or array to a place holder. |
| **lrd_cancel** | Cancels the previous statement. |
| **lrd_data_info** | Gets I/O information. (CtLib only) |
| **lrd_dynamic** | Specifies a dynamic SQL statement to be processed. (CtLib only) |
| **lrd_exec** | Executes the previously specified SQL statement. |
| **lrd_send_data** | Sends data to the server. |
| **lrd_stmt** | Specifies an SQL statement to be processed. |

### Statement Correlating Functions

| | |
|---|---|
| **lrd_save_col** | Saves the value of a table cell to a parameter. |
| **lrd_save_value** | Saves a place holder descriptor value to a parameter. |
| **lrd_save_ret_param** | Saves the value of a return-parameter to a parameter. (CtLib only) |
| **lrd_save_last_rowid** | Saves the last rowid to a parameter (Oracle). |

### Variable Handling Functions

| | |
|---|---|
| **lrd_assign** | Assigns a null-terminated string to a variable. |
| **lrd_assign_ext** | Assigns a storage area to a variable. |
| **lrd_assign_literal** | Assigns a literal string (containing null-characters) to a variable. |
| **lrd_assign_bind** | Assigns a null-terminated string to a variable and binds it to a place holder. |
| **lrd_assign_bind_ext** | Assigns a storage area value to a variable and binds it to a place holder. |
| **lrd_assign_bind_literal** | Assigns a literal string (containing null-characters) to a variable and binds it to a place holder. |
| **lrd_to_printable** | Converts a variable to a printable string. |

### Siebel Functions

| | |
|---|---|
| **lrd_siebel_incr** | Increments a string by a specified value. |
| **lrd_siebel_str2num** | Converts a base 36 string to a base 10 number. |
| **SiebelPostSave_x** | Saves the future values of Siebel parameters. |
| **SiebelPreSave_x** | 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**.

| | |
|---|---|
| **lrd_attr_set** | Sets an attribute for an LRDDBI handle. |
| **lrd_attr_set_from_handle** | Sets an attribute using an LRDDBI handle pointer. |

| | |
|---|---|
| **lrd_attr_set_literal** | Sets an LRDDBI handle attribute using a literal string. |
| **lrd_env_init** | Allocates and initializes an LRDDBI handle. |
| **lrd_handle_alloc** | Explicitly allocates and initializes an LRDDBI handle. |
| **lrd_handle_free** | Explicitly frees an LRDDBI handle. |
| **lrd_initialize_db** | Initializes the database process environment. |
| **lrd_logoff** | Terminates a simple database session. |
| **lrd_logon** | Begins a simple database session. |
| **lrd_logon_ext** | Begins a simple database session (extended). |
| **lrd_oci8_to_oci7** | Converts and Oracle OCI 8 connection to an Oracle OCI 7 connection. |
| **lrd_ora8_attr_set** | Sets an attribute for an LRDDBI handle—shorthand. |
| **lrd_ora8_attr_set_from_handle** | Sets an attribute using an LRDDBI handle pointer. |
| **lrd_ora8_attr_set_literal** | Sets an LRDDBI handle attribute using a literal string—shorthand. |
| **lrd_ora8_bind_col** | Binds a host variable to an output column. |
| **lrd_ora8_bind_placeholder** | Binds a host variable to a placeholder. |
| **lrd_ora8_commit** | Commits the current transaction for an Oracle 8.x client. |
| **lrd_ora8_exec** | Executes an SQL statement in Oracle 8.x. |
| **lrd_ora8_fetch** | Fetches the next row in the result set. |
| **lrd_ora8_handle_alloc** | Explicitly allocates and initializes an LRDDBI handle—shorthand. |
| **lrd_ora8_lob_locator_assign** | Assigns one large object locator to another. |
| **lrd_ora8_lob_locator_temporary** | Creates a temporary large object. |

| | |
|---|---|
| **lrd_ora8_lob_read** | Reads characters from a large object descriptor. |
| **lrd_ora8_lob_write** | Writes characters to a large object descriptor. |
| **lrd_ora8_rollback** | Rolls back the current transaction for an Oracle 8.x client. |
| **lrd_ora8_print** | Prints rows fetched by an Oracle autofetch operation. |
| **lrd_ora8_save_last_rowid** | Saves a rowid to a parameter. |
| **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. |

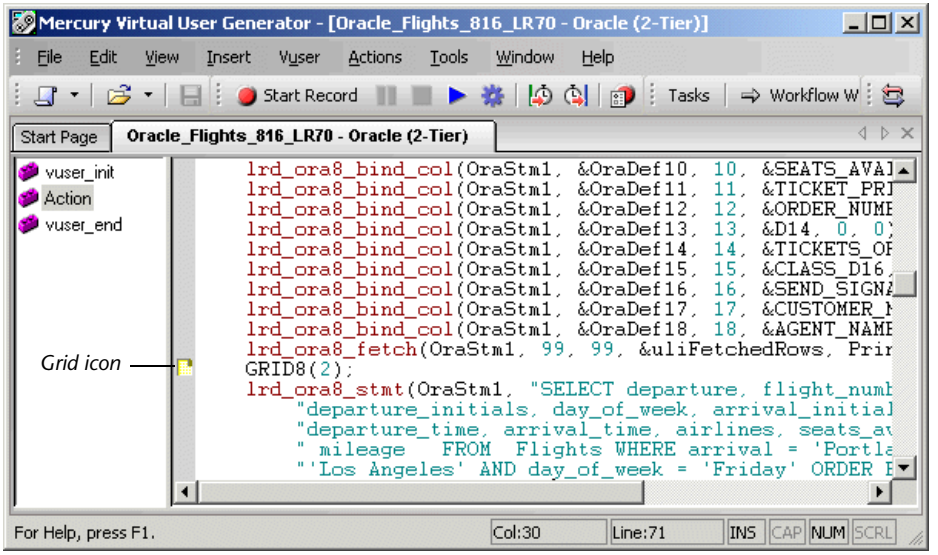
In the following script, VuGen recorded the actions of an operator who opened a connection to an Oracle server and then performed a query requesting the local settings.

```
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);
```

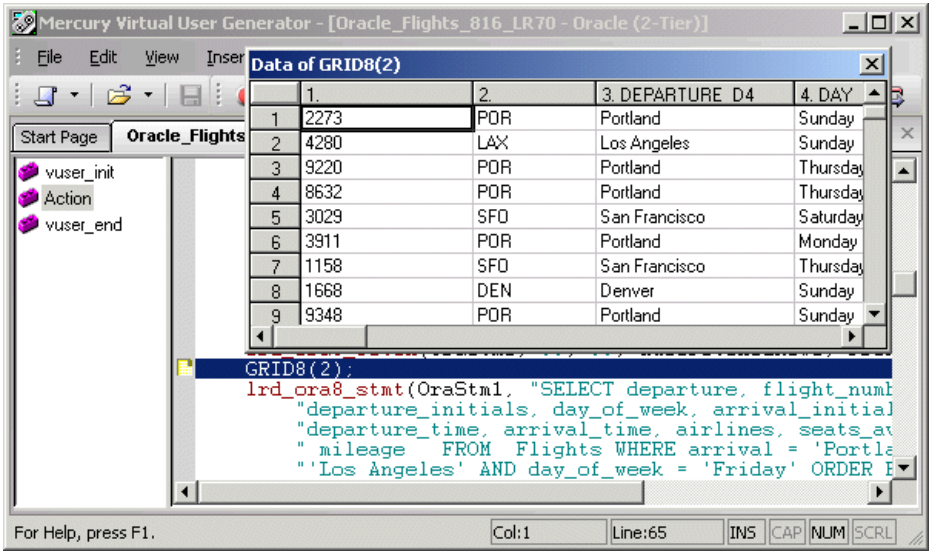
## Grids

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.

The data grid is represented by a **GRID** statement or **GRID8** for Oracle 8.To open the data grid, click on the icon in the margin adjacent to the GRID statement.



In the following example, VuGen displays a grid for a query executed on a flights database. The query retrieves the flight number, airport code, departure city, day of the week, and other flight-relevant information.




The grid columns are adjustable in width. You can scroll up to 100 rows using the scroll bar.

To correlate a value or save the data to a file, click in a cell and use the right-click menu options, **Create Correlation** or **Save To File**.

### Row Information

VuGen generates an **lrd_fetch** function for each SQL query.

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

The second parameter of the function indicates the number of rows fetched. This number can be positive or negative.

### Positive Row Values

A positive value shows the number of rows fetched during recording, and indicates that not all rows were fetched. (For example, if the operator cancelled the query before it was completed.)

In the following example, four rows were retrieved during the database query, but not all of the data was fetched.

```
lrd_fetch(Csr1, 4, 1, 0, PrintRow7, 0);
```

During execution, the script retrieves the number of rows indicated by the positive value (provided the rows exist).

### Negative Row Values

A negative row value indicates that all available rows were fetched during recording. The absolute value of the negative number is the number of rows fetched.

In the following example, all four rows of the result set were retrieved:

```
lrd_fetch(Csr1, -4, 1, 0, PrintRow7, 0);
```

When you execute an **lrd_fetch** statement containing a negative row value, it retrieves all of the available rows in the table at the time of the run—not necessarily the number at the time of recording. In the above example, all four rows of the table were retrieved during the recording session. However, if more rows are available during script execution, they are all retrieved.

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

# Evaluating Error Codes

When a Vuser executes an LRD function, the function generates a return code. A return code of 0 indicates that the function succeeded. For example, a return code of 0 indicates that another row is available from the result set. If an error occurs, the return code indicates the type of error. For example, a return code of 2014 indicates that an error occurred in the initialization.

There are four types of return codes, each represented by a numerical range:

| Type of Return Code | Range |
|---|---|
| Informational | 0 to 999 |
| Warning | 1000 to 1999 |
| Error | 2000 to 2999 |
| Internal Error | 5000 to 5999 |

For more detailed information on the return codes, 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 in the following ways:

- Globally Modifying Error Handling—to the entire script, or to a segment of the script
- Locally Modifying Error Handling—to a specific function only

## Globally Modifying Error Handling

You can change the way that Vusers handle errors by issuing an LRD_ON_ERROR_CONTINUE or LRD_ON_ERROR_EXIT statement. By default, a Vuser aborts the script execution when it encounters any type of error—database, parameter related, and so on. To change the default behavior, insert the following line into your script:

```
LRD_ON_ERROR_CONTINUE;
```

From this point on, the Vuser continues script execution, even when an error occurs.

You can also specify that the Vuser continue script execution when an error occurs only within a segment of the script. For example, the following code tells the Vuser to continue script execution even if an error occurs in the **lrd_stmt** or **lrd_exec** functions:

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

Use the LRD_ON_ERROR_CONTINUE statement with caution, as significant and severe errors may be missed.

### Locally Modifying Error Handling

You can set error handling for a specific function by modifying the severity level. Functions such as **lrd_stmt** and **lrd_exec**, which perform database operations, use severity levels. The severity level is indicated by the function's final parameter, **miDBErrorSeverity**. This parameter tells the Vuser whether or not to continue script execution when a database error occurs (error code 2009). The default, 0, indicates that the Vuser should abort the script when an error occurs.

For example, if 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.

# 24

# 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:

- About Correlating Database Vuser Scripts
- Scanning a Script for Correlations
- Correlating a Known Value
- Database Correlation Functions

***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. Correlating the query means that you save a run-time value to a parameter. You then use the saved value at a later point in the same script. In summary, correlation is using the results of one statement as input to another.

There are many queries whose inputs depend on the result of prior queries. To emulate this behavior, use VuGen's correlation capabilities.

# Scanning a Script for Correlations

VuGen provides a correlation utility to help you repair your script and allow a successful replay. It performs the following steps:

- Scans for potential correlations
- Inserts the appropriate correlation function to save the results to a parameter
- Replaces the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

**To scan and correlate a script detected with automatic correlation:**

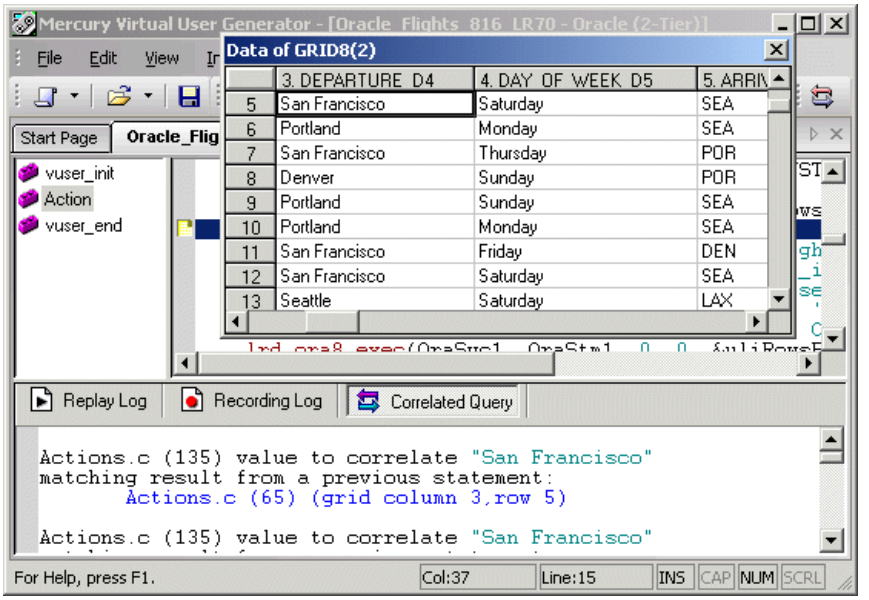
**1** Open the Output window.

Select **View** > **Output** to display the output tabs at the bottom of the window. Check for errors in the Replay Log tab. Often, these errors can be corrected by correlation.

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

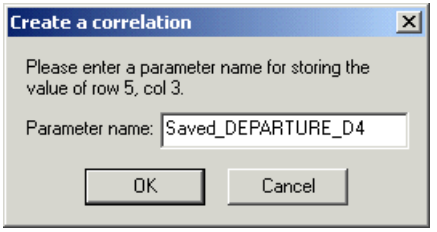
VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.

In the following example, in the **lrd_ora8_stmt** statement, VuGen detected a value to correlate.




**3** In the Correlated Query tab, double-click on the result you want to correlate. Click on the words **(grid column x, row y)** VuGen sends the cursor to the location of the value in your grid.

**4** Choose **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.




**5** Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**, **lrd_ora8_save_col**) which saves the result to a parameter.

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

7 A message box opens asking if you want to search for all occurrences of the value in the script.
To replace only the value in the selected statement, click **No**.
To search and replace additional occurrences, click **Yes**.

8 A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

9 Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. Note that if you 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.

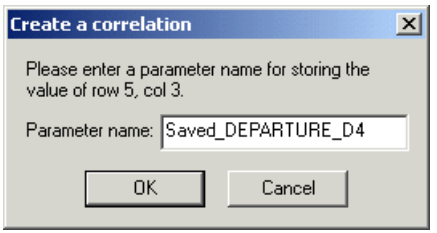
**To correlate a specific value:**

1 Locate the statement in your script, with the query containing the value you want to correlate. This is usually one of the arguments of the **lrd_assign**, **lrd_assign_bind**, or **lrd_stmt** functions. Select the value without the quotation marks.

2 Choose **Scan for Correlations (at cursor)** from the right-click menu. VuGen scans the selected value for correlations.

3 In the Output window's **Correlated Query** tab, double-click on the result you want to correlate. Click on the words (**grid column x, row y**). VuGen sends the cursor to the location of the value in your grid.

**4** In the grid, click on the value you want to correlate and choose **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.




**5** Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrd_save_value**, **lrd_save_col**, or **lrd_save_ret_param**, **lrd_ora8_save_col**) which saves the result to a parameter.

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

**7** A message box opens asking if you want to search for all occurrences of the value in the script.
To replace only the value in the selected statement, click **No**.
To search and replace additional occurrences, click **Yes**.

**8** A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

**9** Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. If you cancel the correlation, VuGen also erases the statement created in the previous step.

---

**Note:** If you are correlating a value from an **lrd_stmt** function, the following data types are not supported: date, time, and binary (RAW, VARRAW).

---

## Database Correlation Functions

When working with Database Vuser scripts, (DbLib, CtLib, Oracle, Informix, and so forth) you can use VuGen's automated correlation feature to insert the appropriate functions into your script. The correlating functions are:

- **lrd_save_col** saves a query result appearing in a grid, to a parameter. This function is placed before fetching the data. It assigns the value retrieved by the subsequent **lrd_fetch** to the specified parameter. (**lrd_ora8_save_col** for Oracle 8 and higher)
- **lrd_save_value** saves the current value of a placeholder descriptor to a parameter. It is used with database functions that set output placeholders (such as certain stored procedures under Oracle).
- **lrd_save_ret_param** saves a stored procedure's return value to a parameter. It is used primarily with database procedures stored in DbLib that generate return values.

---

**Note:** VuGen does not apply correlation if the saved value is invalid or NULL (no rows returned).

---

For more information about these functions and their arguments, refer to the *Online Function Reference*.

# 25

# Developing DNS Vuser Scripts

VuGen allows you to emulate network activity by directly accessing a DNS server.

This chapter describes:

- About Developing DNS Vuser Scripts
- 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.

To create a script for the DNS protocol, choose **File > New** to open the New Virtual User dialog box. Choose the Domain Name Resolution (DNS) protocol type from the Client/Server category. Since recording is not supported for DNS, you program the script with the appropriate DNS, Vuser API and C functions. For more information on these functions, refer to the *Online Function Reference* (**Help > Function Reference**).

After you create a Vuser script, you integrate it into a session step or scenario on either a Windows or UNIX platform. For more information on integrating Vuser scripts in a session step or scenario, refer to the *LoadRunner Controller User's Guide* or the *Tuning Module 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, refer to 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;
}
```

# 26

# 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:

- About Recording Windows Sockets Vuser Scripts
- Getting Started with Windows Sockets Vuser Scripts
- Setting the WinSock Recording Options
- Using LRS Functions

***The following information applies to all protocols recorded on a Windows Sockets level, including the Web/Winsock Dual Protocol.***

## About Recording Windows Sockets Vuser Scripts

The Windows Sockets protocol is ideal for analyzing the low level code of an application. For example, to check your network, you can use a Windows Sockets (WinSock) script to see the actual data sent and received by the buffers. The WinSock type can also be used for recording other low level communication sessions. In addition, you can record and replay applications that are not supported by any of the other Vuser types.

When you record an application which uses the Windows Sockets protocol, VuGen generates functions that describe the recorded actions. Each function begins with an **lrs** prefix. The LRS functions relate to the sockets, data buffers, and the Windows Sockets environment. Using VuGen, you record your application's API calls to the Winsock.dll or Wsock32.dll.

For example, you could create a script by recording the actions of a *telnet* application.

In the following example, **lrs_send** sends data to a specified socket:

```
lrs_send("socket22", "buf44", LrsLastArg);
```

You can view and edit the recorded script from VuGen's main window. The Windows Sockets API calls that were recorded during the session are displayed in the window, allowing you to track your network activities.

VuGen can display a WinSock script in two ways:

- As an icon-based representation of the script. This is the default view, and is known as the **Tree view**.
- As a text-based representation of the script showing the Windows Sockets API calls. This is known as the **Script view**.

You use VuGen to view and edit your Vuser script from either the Tree view or Script view. For more information, see "Viewing and Modifying Vuser Scripts" on page 17.

After creating a script, you can view the recorded data as a snapshot or as a raw data file. For details, see Chapter 27, "Working with Windows Socket Data."

## 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 4, "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 7, "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 8, "Working with VuGen Parameters."

**4 Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 11, "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 12, "Configuring Run-Time Settings." and Chapter 13, "Configuring Network Run-Time Settings."

**6 Run the Vuser script from VuGen.**

Save and run the Vuser script from VuGen to verify that it runs correctly.

For details, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

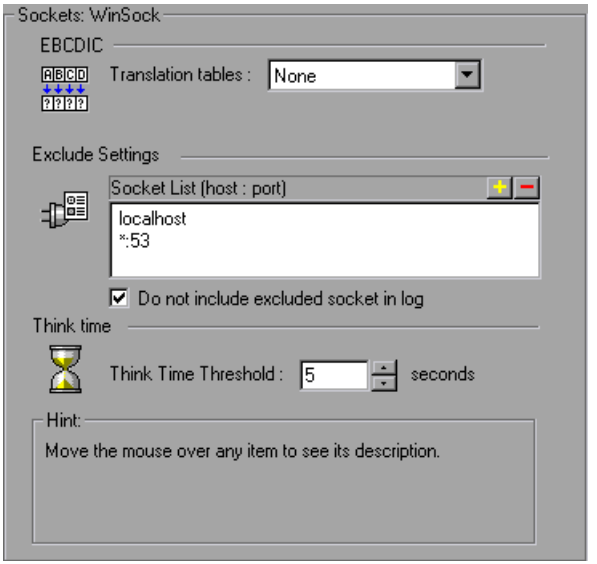
After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# Setting the WinSock Recording Options

The following recording options are available for WinSock Vusers:

- Configuring the Translation Table
- Excluding Sockets
- Setting the Think Time Threshold

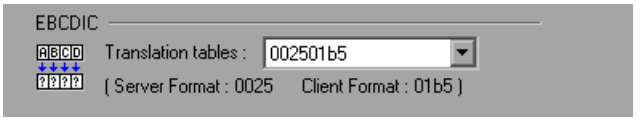
To open the Recording Options dialog box, choose **Tools** > **Recording Options** or click the **Options** button in the Start Recording dialog box. VuGen displays the WinSock options.



## Configuring the Translation Table

To display data in EBCDIC format, you specify a translation table in the recording options.

The Translation Table lets you specify the format for recording sessions. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. 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 VuGen's installation directory. If your system uses different translation tables, copy them to the **ebcdic** directory.

---

**Note:** If your data is in ASCII format, it does not require translation. You must select the **None** option, the default value. If you do select a translation table, VuGen will translate the ASCII data.

---

When working on Solaris machines, you must set the following environment variables: on all machines running the Vuser scripts.

```
setenv LRSDRV_SERVER_FORMAT 0025
setenv LRSDRV_CLIENT_FORMAT 04e4
```

## Excluding Sockets

VuGen supports the Exclude Socket feature, allowing you to exclude a specific socket from your recording session. To exclude all actions on a socket from your script, you specify the socket address in the Exclude Socket list. To add a socket to the list, click the plus sign in the upper right corner of the box and enter the socket address in one of the following formats:

| Value | Meaning |
|---|---|
| host:port | Exclude only the specified port on the specified host. |
| host | Exclude all ports for the specified host. |
| :port | Exclude the specified port number on the local host. |
| *:port | Exclude the specified port number on all hosts. |

You can exclude multiple hosts and ports by adding them to the list. To remove a socket from the excluded list, select the socket address and click the minus sign in the upper right corner of the box. 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 **lrs** prefix. VuGen automatically records most of the LRS functions listed in this section during a Windows Sockets session. You can also manually program any of the functions into your Vuser script. For more information about the LRS functions, refer to the *Online Function Reference* (**Help > Function Reference**).

### Socket Functions

| | | |
|---|---|---|
|  | **lrs_accept_connection** | Accepts a connection on a listening socket. |
|  | **lrs_close_socket** | Closes an open socket. |
|  | **lrs_create_socket** | Initializes a socket. |
|  | **lrs_disable_socket** | Disables an operation on a socket. |
|  | **lrs_exclude_socket** | Excludes a socket during replay. |
|  | **lrs_get_socket_attrib** | Gets a socket's attributes. |
|  | **lrs_get_socket_handler** | Gets a socket handler for the specified socket. |

| | |
|---|---|
|  **lrs_length_receive** | Receives data from a buffer of a specified length. |
|  **lrs_receive** | Receives data from a socket. |
|  **lrs_receive_ex** | Receives data of a specific length from a datagram or stream socket. |
|  **lrs_send** | Sends data on a datagram or to a stream socket. |
|  **lrs_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_recv_timeout** | Sets a timeout for receiving the initial expected data on a socket. |
|  | **lrs_set_recv_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.

# 27

# Working with Windows Socket Data

After you record a session in the Windows Socket protocol you can view and manipulate the data.

This chapter describes:

- About Working with Windows Socket Data
- Viewing Data in the Snapshot Window
- Navigating Through the Data
- Modifying Buffer Data
- Modifying Buffer Names
- Viewing Windows Socket Data in Script View
- 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 VuGen, you have multiple data buffers containing the data.

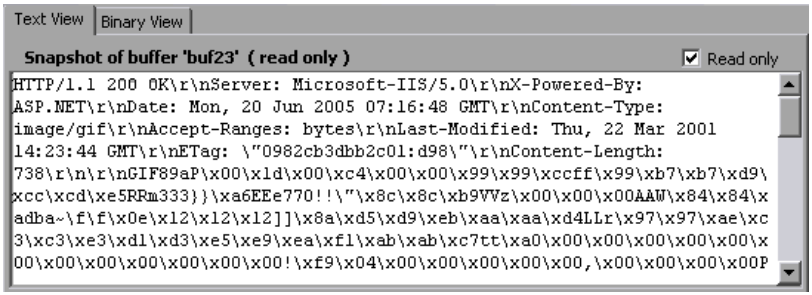
When you view the WinSocket script in tree view, VuGen provides a snapshot window which allows you to navigate within the data buffers and modify the data.

When working in script view, you can view the raw data in the **data.ws** file. For more information, see "Viewing Windows Socket Data in Script View" on page 402.

## Viewing Data in the Snapshot Window

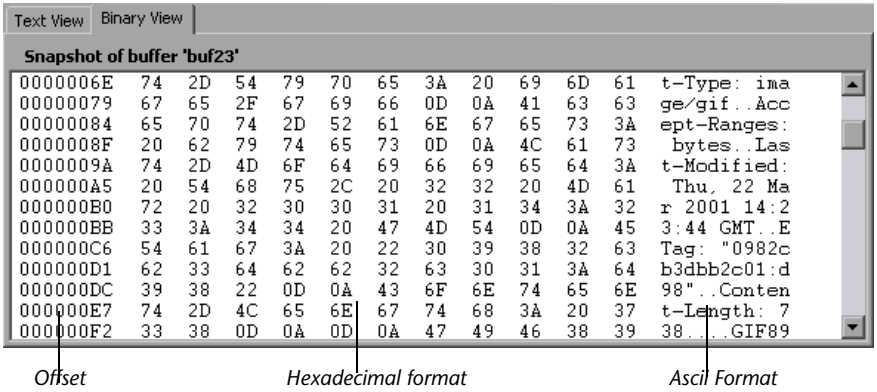
When viewing a Windows Socket script in tree view, VuGen provides a buffer snapshot window which displays the data in an editable window. You can view a snapshot in either **Text** view or **Binary** view.

The text view shows a snapshot of the buffer with the contents represented as text.



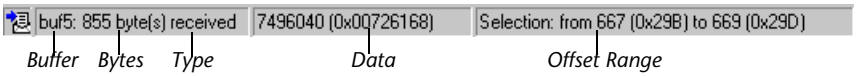

By default, VuGen stores the buffer data as read-only data. If you want to modify the contents of the buffer, clear the **Read only** box in the buffer's Text View. VuGen issues a warning that bookmarks and parameters may be affected.

The binary view shows the data in hexadecimal representation. The left column shows the offset of the first character in that row.
The middle column displays the hexadecimal values of the data. The right column shows the data in ASCII format.




The status bar below the buffer snapshot provides information about the data and buffer:

- **Buffer number:** The buffer number of the selected buffer.
- **Total bytes:** the total number of bytes in the buffer.
- **Buffer type:** the type of buffer—received or sent.
- **Data:** the value of the data at the cursor in decimal and hexadecimal formats, in Little Endian order (reverse of how it appears in the buffer).
- **Offset:** the offset of the selection (or cursor in text view) from the beginning of the buffer. If you select multiple bytes, it indicates the range of the selection.




The status bar also indicates whether or not the original data was modified.



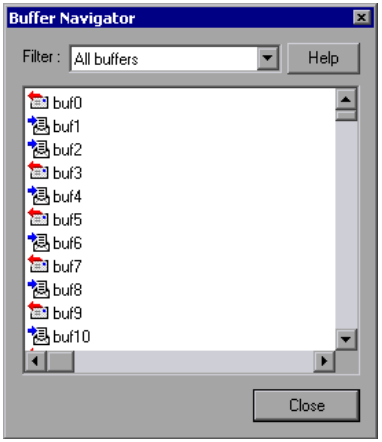
# Navigating Through the Data

In tree view, VuGen provides several tools that allow you to navigate through the data in order to identify and analyze a specific value:

- Buffer Navigator
- Go To Offset
- Bookmarks

## Buffer Navigator

By default, VuGen displays all the steps and buffers in the left pane. The Buffer Navigator is a floating window that lets you display only the receive and send buffers steps (**lrs_send**, **lrs_receive**, **lrs_receive_ex**, and **lrs_length_receive**). In addition, you can apply a filter and view either the send or receive buffers.




When you select a buffer in the navigator, its contents are displayed in the buffer snapshot window.

If you change a buffer's name after recording, its contents will not appear in the snapshot window when you click on the step. To view the renamed buffer's data, use the buffer navigator and select the new buffer's name. VuGen issues a warning message indicating that parameter creation will be disabled for the selected buffer.

To open the Buffer Navigator, choose **View** > **Buffer Navigator**. To close the navigator, click the X in the top right corner of the navigator dialog box.

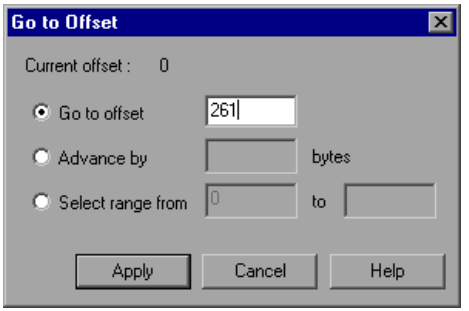
Note that you can also navigate between buffers by clicking on the buffer step in the left pane's tree view. The advantages of the buffer navigator are that it is a floating window with filtering capabilities.

## 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:**

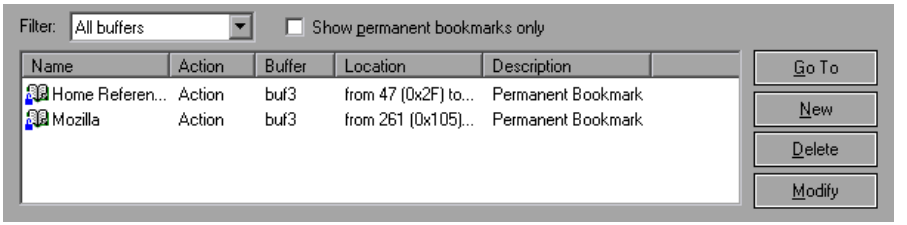
1. Click within the snapshot window. Then select **Go to offset** from the right-click menu. The Go to offset dialog box opens.




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 binary view the bookmark block is marked in red. Also in binary view, when you place your cursor over a bookmark, a popup text box opens indicating the name of the bookmark.

You can create both permanent and simple bookmarks. A permanent bookmark is always marked within the buffer's binary view—it is enclosed by a blue box. The bookmark stays selected in blue, even when pointing to another location in the buffer. The cursor location is marked in red. A simple bookmark, however, is not permanently marked. When you jump to a simple bookmark, it is marked in red, but once you move the cursor within the buffer, the bookmark is no longer selected. By default bookmarks are permanent.

**To work with bookmarks:**

1. To create a bookmark, select one or more bytes in a buffer snapshot (text or binary view) and select **New Bookmark** from the right-click menu.
2. To view the bookmark list, 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 form being Permanent to simple (permanent means that it is always marked, even when you move the cursor to a new location), select the bookmark, perform a right-click, and clear the check adjacent to **Permanent Bookmark**.

6 To display only permanent bookmarks in the list, select the **Show Permanent Bookmarks only** check box in the **Bookmarks** tab.

7 To view bookmarks from a specific buffer, select a bookmark from the desired buffer and 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.

- Inserting Data
- Editing Data
- Parameterizing the Data

### Inserting Data

You can insert a numerical value into a data buffer. You can insert it as a single, double-byte or 4-byte value.

**To insert a number into a data buffer:**

1 Click at a location in the buffer.

2 Open the right-click menu and choose **Advanced** > **Insert Number** > **Specify...**

3 Enter the ASCII value that you want to insert into the **Value** box.

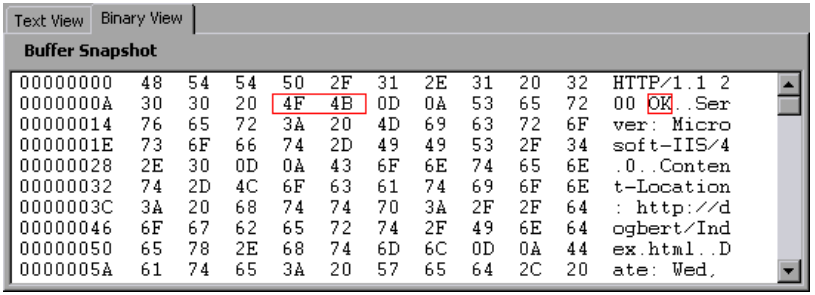
**4** Select the size of the data you want to insert: 1 byte, 2 bytes, or 4 bytes from the **Size** box.

**5** Click **OK** to finish. VuGen inserts the hexadecimal representation of the data into the buffer.

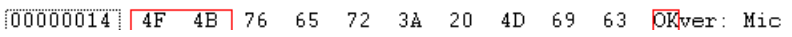
### Editing Data

You can perform all of the standard edit operation on buffer data: copy, paste, cut, delete, and undo. In the binary view you can specify the actual data to insert. VuGen allows you to specify the format of the data—single byte, 2-byte, or 4-byte, and hexadecimal or decimal value. You can copy binary data and insert it as a number into the buffer. You can see the decimal or hexadecimal numbers in the right column of the binary view.

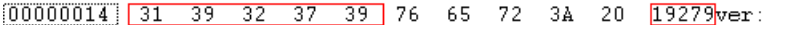
In the following example, the word OK was selected.



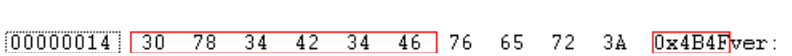

If you perform simple copy (CTRL+C) and paste (CTRL+V) operations at the beginning of the next line of data, it inserts the actual text.




If you choose and **Advanced Copy as Number** > **Decimal** and then paste the data, it inserts the decimal value of the ASCII code of the selected characters:




If you choose and **Advanced Copy as Number** > **Hexadecimal** and then paste the data, it inserts the hexadecimal value of the ASCII code of the selected characters:



The Undo Buffer retains all of the modifications to the buffer. This information is saved with the file—if you close the file it will still be available. If you want to prevent others from undoing your changes, you can empty the Undo buffer. To empty the Undo buffer, 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

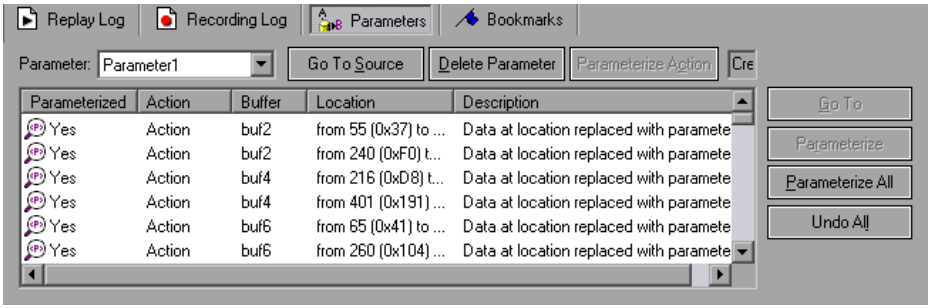
In tree view, VuGen lets you parameterize the data directly from the buffer snapshot view. You can specify a range of what to parameterize and you can specify borders. If you do not specify borders for the parameterized string, then VuGen inserts an **lrs_save_param** function into your script. If you 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 8, "Working with VuGen Parameters."

The next sections discuss the correlation of data in Receive buffers.

After you create a parameter, VuGen lists all the locations in which it replaced the string with a parameter. VuGen also provides information about the creation of the parameter—the buffer in which it was created and the offset within the buffer. It lists all occurrences of the parameter in the Output Window's **Parameters** tab, below the snapshot view.




VuGen allows you to manipulate the parameters:

**Filtering:** You can filter the parameter replacements by the parameter name.

**Go to Source:** Select a replacement and click Go To Source to jump to the exact location of the replaced parameter within the buffer.

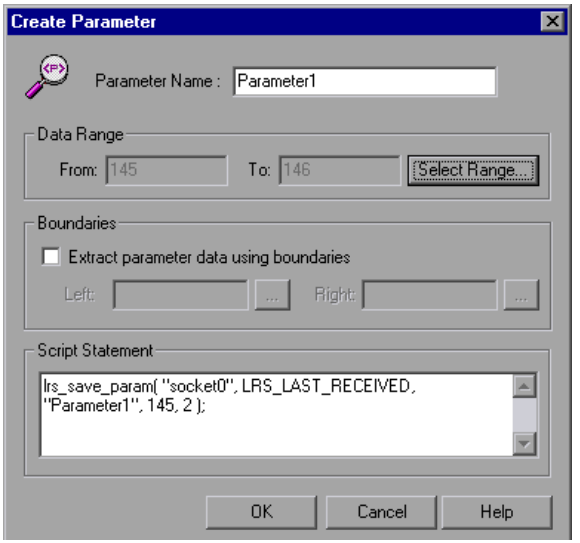
**Deleting:** You can delete any one of the parameters. When you delete a parameter, VuGen replaces the data with its original value and removes the parameterization function from the script.

**Name:** You can provide a name to each replacement.

**Undo Replacement:** You can also undo one or more replacements displayed in the list.

**To parameterize data from the snapshot window:**

1 Select the data you want to parameterize and 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.

To specify boundaries:

➤ Select the **Extract Parameter Data Using Boundaries** check box. VuGen changes the function in the **Script Statement** section from **lrs_save_param** to **lrs_save_searched_string**. Click **Done.**

➤ Click the browse button adjacent to the **Left** box in the **Boundaries** section. A small dialog box opens, indicating your selection within the buffer. Select the boundaries within the buffer and click **Done**. Repeat this step for the right boundary.

**5** Make the desired modifications to the arguments in the **Script Statement** section. For example you can add **_ex** to the **lrs_save_param** function to specify an encoding type. For more information about these functions refer to the *Online Function Reference*.

**6** Click **OK** to create the parameter. VuGen asks you for a confirmation before replacing the parameter. Click **Yes**. You can view all the replacements in the **Parameters** tab.

**7** To jump to the original location of the parameter within its buffer, select it and click **Go To Source**.

**8** To jump to the buffer location of the selected replacement, select it and click **Go To**.

**9** To delete an entire parameter, choose the parameter in the **Filter** box and click **Delete Parameter**.

**10** To undo a replacement, select it in the **Parameters** tab and click **Undo**. To undo all replacements of the displayed parameter, select it in the **Parameters** tab and click **Undo All.**

**11** When you undo specific replacements, the label changes from **Replaced** to **Found**. To reapply the parameterization rule to those that were undone, click **Replace** or **Replace All**.

**12** To delete an entire parameter and undo all the replacements, select the parameter in the **Filter** box and click **Delete Parameters**.

**13** 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.

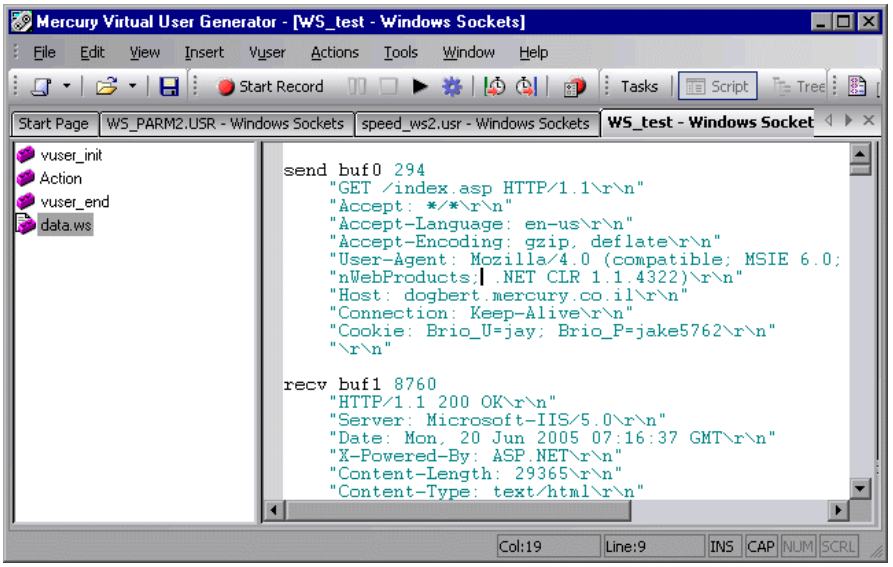
If you created parameters in the buffer and it is not longer available, VuGen prompts you to delete the parameters from the buffer in which they were defined. When you delete the parameter, all replacements are undone, even those in other buffers.

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:

recv buf*index*  *number of bytes received*
send buf*index*

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, the descriptor is followed by the actual ASCII data that was transferred by the sockets.

If your data is in EBCDIC format, it must be translated through a look-up table. For information on setting the translation table, see “Setting the WinSock Recording Options” on page 382. 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"

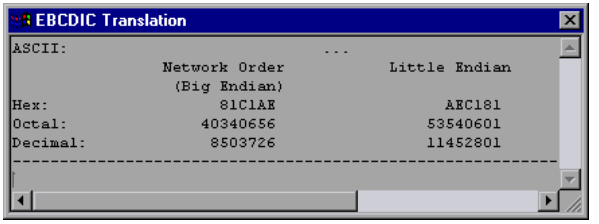
recv buf1 15
	"\xff\xfd\x18\xff\xfd\x1f\xff\xfd"
	"#"
	"\xff\xfd"
	"'"
	"\xff\xfd"
	"$"

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.

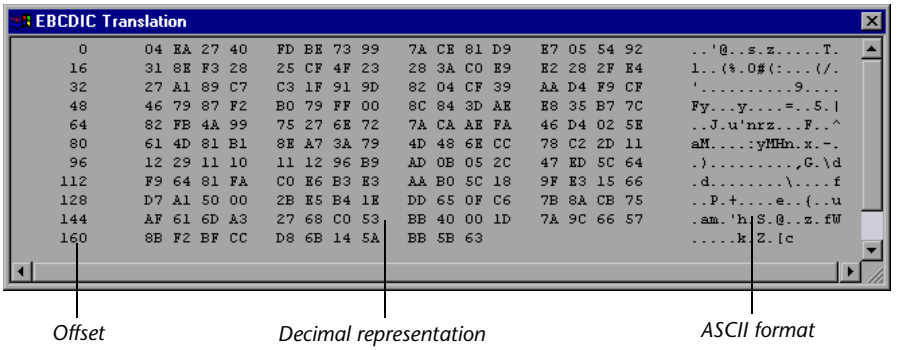
To display the data in the viewer window, select the data and press F7. If the selected text is less than four characters, VuGen displays the data in **short format**, showing the hexadecimal, decimal and octal representations.




You can customize the short format in the **conv_frm.dat** file as described in “Setting the Display Format” on page 408.

If the selected text is more than four characters, VuGen displays the data in several columns in **long format**. You can customize the long format by modifying the **conv_frm.dat** file, as described in “Setting the Display Format” on page 408.

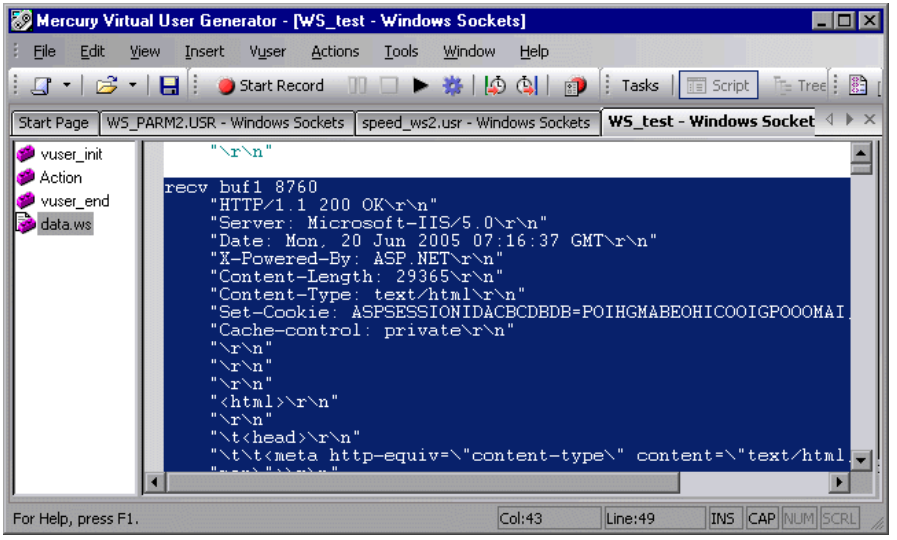
In the default format, the first column displays the character offsets from the beginning of the marked section. The second column displays the hexadecimal representation of the data. The third column shows the data in ASCII format. When displaying EBCDIC data, all non-printable ASCII characters (such as /n), are represented by dots.



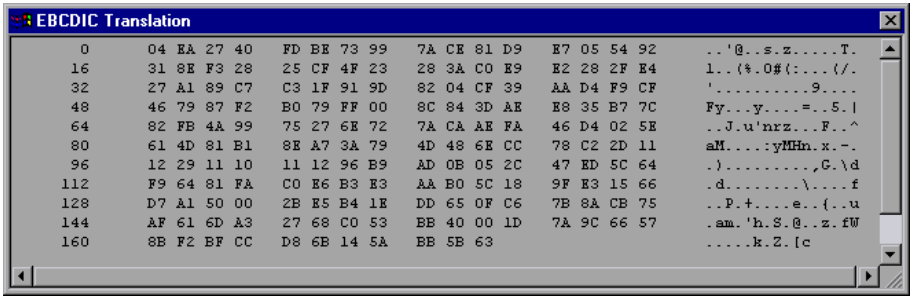
The F7 viewer utility is especially useful for parameterization. It allows you to determine the offset of the data that you want to save to a parameter.

**To determine the offset of a specific character:**

**1** View **data.ws** and select the data from the beginning of the buffer.




**2** Press F7 to display the data and the character offsets. Since more than four characters were selected, the data is displayed in long format.




**3** Locate the value you want to correlate in the ASCII data. In this example, we will correlate the number 13546 (a process ID during a UNIX session) which begins at the 31st character—the last character in the second line.

**4** Use the offset value in the **lrs_save_param_ex** function in order to correlate the value of the process ID. For more information, see Chapter 11, "Correlating Statements."

## Setting the Display Format

You can specify how VuGen will display the buffer data in the viewer (F7) window. The **conv_frm.dat** file in the **lrun/dat** directory contains the following display parameters:

**LongBufferFormat:** The format used to display five or more characters. Use nn for offset, XX for the hex data, and aa for ASCII data.

**LongBufferHeader:** A header to precede each buffer in Long buffer format.

**LongBufferFooter:** A footer to follow each buffer in Long buffer format.

**ShortBufferFormat:** The format used to display four characters or less. You can use standard escape sequences and conversion characters.

The supported escape sequence characters are:

| | |
|---|---|
| \a | Bell (alert) |
| \b | Backspace |
| \f | Formfeed |
| \n | New line |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \' | Single quotation mark |
| \" | Double quotation mark |
| \\ | Backslash |
| \? | Literal question mark |
| \ooo | ASCII character -octal |

The supported conversion characters are:

| | |
|---|---|
| %a | ASCII representation |
| %BX | Big Endian (Network Order) Hex |
| %BO | Big Endian (Network Order) Octal |
| %BD | Big Endian (Network Order) Decimal |
| %LX | Little Endian Hex |
| %LO | Little Endian Octal |
| %LD | Little Endian Decimal |

**AnyBufferHeader:** A header to precede each buffer.

**AnyBufferFooter:** A footer to follow each buffer.

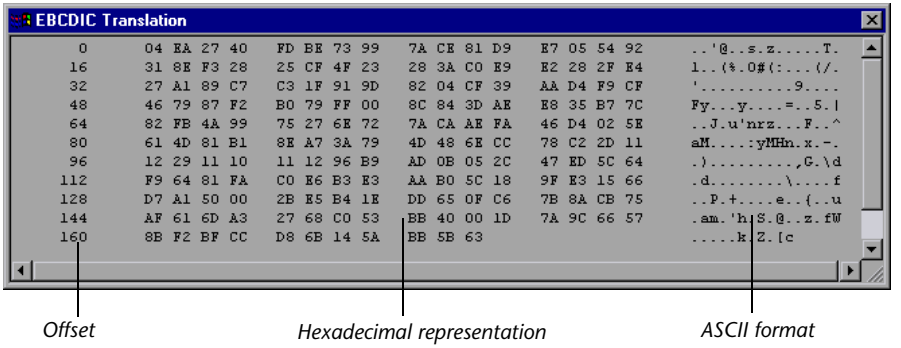
**NonPrintableChar:** The character with which to represent non-printable ASCII characters.

**PrintAllAscii:** Set to 1 to force the printing of non-printable ASCII characters.

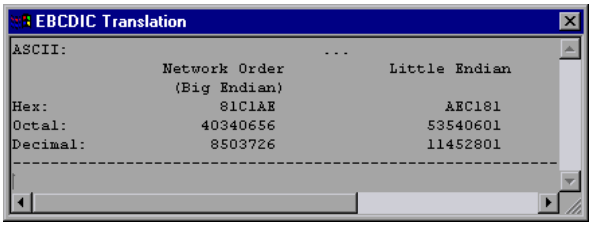
In the default settings, long and short formats are set, and a dot is specified for non-printable characters.

```
[BufferFormats]
LongBufferFormat=nnnnnnnn     XX XX XX XX   XX XX XX XX   XX XX XX
XX   XX XX XX XX     aaaaaaaaaaaaaaaa\r\n
LongBufferHeader=
LongBufferFooter=
ShortBufferFormat=ASCII:\t\t\t%a\r\n\t\tNetwork Order\t\tLittle
Endian\r\n\t\t (Big Edian)\r\nHex:\t\t%BX\t\t%LX\r\nOc-
tal:\t\t%BO\t\t%LO\r\nDecimal:\t%BD\t\t%LD\r\n
AnyBufferHeader=
AnyBufferFooter=----------------------------------------------------------------------\r\n
NonPrintableChar=.
PrintAllAscii=0
```

The default LongBufferFormat is displayed as:




The default ShortBufferFormat is displayed as:



# Debugging Tips

VuGen offers several means which allow you to debug your script. You can view the various output logs and windows for detailed messages issued during execution.

Specifically for Windows Sockets Vuser scripts, VuGen provides additional information about buffer mismatches. A buffer mismatch indicates a variation in the received buffer size (generated during replay) and the expected buffer (generated during record). However, if the received and expected buffer are the same size, even though the contents are different, a mismatch message is not issued. This information can help you locate a problem within your system, or with your Vuser script.

You can view the buffer mismatch information in the Execution log. 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, refer to the *Online Function Reference*.

**To correlate the WinSock Vuser statements:**

**1** Insert the **lrs_save_param_ex** statement into your script at the point where you want to save the buffer contents. You can save user, static, or received type buffers.

```
lrs_save_param_ex (socket, type, buffer, offset, length, encoding, parameter);
```

**2** Reference the parameter.

View the buffer contents by selecting the **data.ws** file in the Data Files box of the main VuGen window. Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in angle brackets (< >).

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

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

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

During execution, the PID of the procedure is different (UNIX assigns unique PIDs for every execution), so killing the recorded PID will be ineffective. To overcome this problem, use **lrs_save_param_ex** to save the current PID to a parameter. Replace the constant with the parameter.

**3** In the **data.ws** file, determine the buffer in which the data was received, buf47.

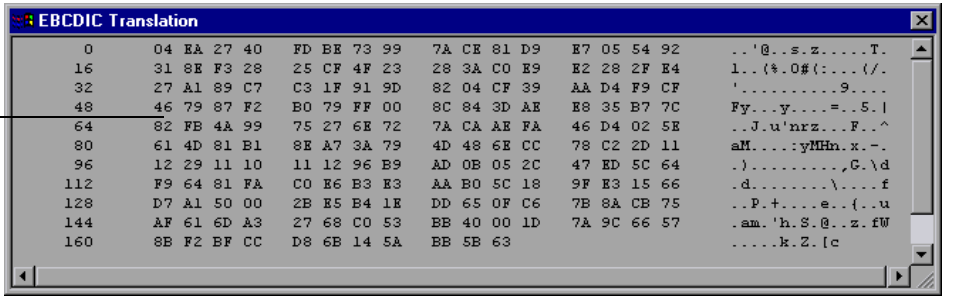
```
recv buf47 98
      "\r"
      "\x00"
      "\r\n"
      "   PID TTY      TIME CMD\r\n"
      " 14602 pts/18   0:00 clock\r\n"
      " 14569 pts/18   0:02 tcsh\r\n"
      "frodo:/u/jay>"
.
.
.
send buf58
      "kill 14602"
```

**4** In the **Actions** section, determine the socket used by buf47. In this example it is **socket1**.

```
lrs_receive("socket1", "buf47", LrsLastArg);
```

**5** Determine the offset and length of the data string to save. Highlight the entire buffer and press F7. The offset of the **PID** is 11 and its length is 5 bytes. For additional information about displaying the data, see "Understanding the Data File Format" on page 404.

*offset of first character in line*




**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 Vuser Scripts

# 28

# Creating Custom Vuser Scripts

In addition to recording a session, you can create a custom Vuser script. You can use both Vuser API functions and standard C, Java, VB, VBScript, or Javascript code.

This chapter describes:

- About Creating Custom Vuser Scripts
- C Vusers
- Using the Workflow Wizard for C Vuser Scripts
- 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 Vuser API or standard programming functions. Vuser API functions allow you to gather information about Vusers. For example, you can use Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test or monitoring.

This chapter describes how to program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes.

You can also develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API function libraries. For more information, see Chapter 76, "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()
{

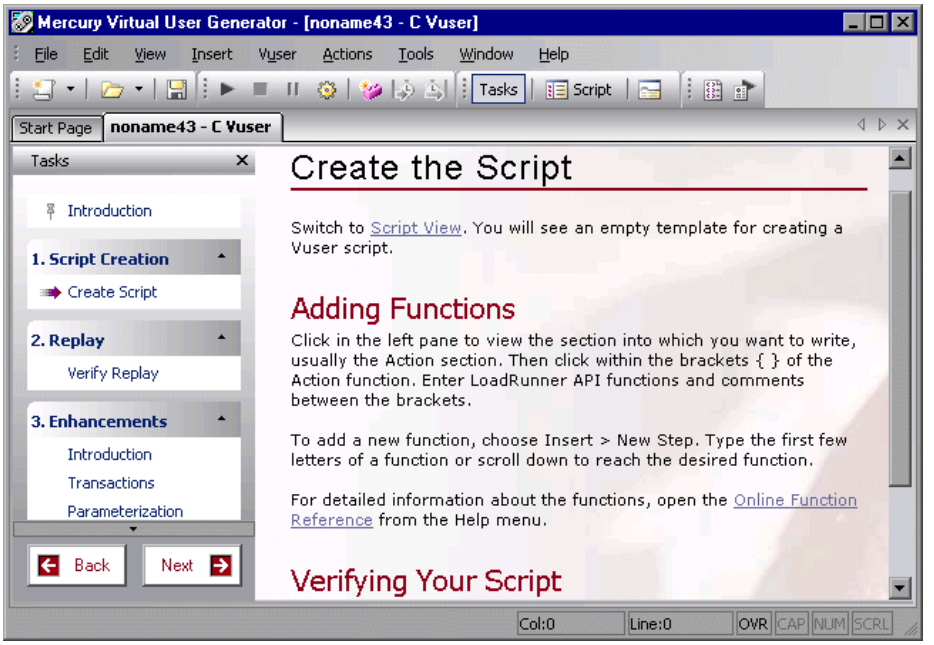
	return 0;
}
```

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.

# Using the Workflow Wizard for C Vuser Scripts

The Workflow Wizard guides you through the steps of creating a script. By clicking on a link in the Tasks pane, you can read about the steps in creating a script, and view information about your replay. Use the **Back** and **Next** buttons to navigate between screens.




If you do not see the Workflow Wizard, make sure that the Tasks pane is open. (You show and hide the Task pane using the **Tasks** button on the toolbar). Then click the first link, **Introduction**.

For more information about the wizard, see Chapter 3, “Using the Workflow Wizard.”

### Create the Script

The Create Script window contains several guidelines for creating a Web Services script.

- **Adding Functions**- describes how and where to enter the functions.
- **Verifying Your Script**- describes how to verify your script after adding functions.

### Guidelines for Using C Functions

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

The C interpreter that is used to run Vuser scripts accepts the standard ANSI C language. It does not support any Microsoft extensions to ANSI C.

Before you add any C functions to a Vuser script, note the following limitations:

- A Vuser script cannot pass the address of one of its functions as a callback to a library function.
- The **stdargs**, **longjmp**, and **alloca** functions are not supported in Vuser scripts.
- Vuser scripts do not support structure arguments or return types. Pointers to structures are supported.
- In Vuser scripts, string literals are read-only. Any attempt to write to a string literal generates an access violation.
- C Functions that do not return int, must be casted. For example, `extern char * strtok();`

#### Calling libc Functions

In a Vuser script, you can call **libc** functions. However, since the interpreter that is used to run Vuser scripts does not support any Microsoft extensions to ANSI C, you cannot use Microsoft's include files. You can either write your own prototypes, or ask 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 lrapi.lr;

public class Actions
{

    public int init() {
        return 0;
    }


    public int action() {
        return 0;
    }


    public int end() {
        return 0;
    }
}
```

Note that for Java type Vusers, you can only edit the **Actions** class. Within the Actions class, there are three methods: **init**, **action**, and **end**. Place initialization code in the **init** method, business processes in the **action** method, and cleanup code in the **end** method.

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 Visual Basic code. This script type lets you incorporate your Visual Basic application into VuGen. 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 Vusers 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 VuGen. 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 the Vuser API functions and VB Script. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
Set lr = CreateObject("LoadRunner.LrApi")
```

# JavaScript Vusers

You can create an empty JavaScript Vuser script, in which to place JavaScript code. This script type lets you incorporate your existing javascript application into VuGen. To create an empty JavaScript Vuser script, 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 the Vuser API functions and the Javascript. For example, for LoadRunner, the following code creates the standard object, **lr**:

```
var lr = new ActiveXObject("LoadRunner.LrApi")
```

# 29

# Programming Java Scripts

VuGen supports Java type users on a protocol level. This chapter explains how to create a Java Vuser script by programming. For information on creating a Java Vuser script through recording, see the chapters describing 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:

- About Programming Java Scripts
- Creating a Java Vuser
- Editing a Java Vuser Script
- Java Vuser API Functions
- 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 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.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Creating a Java Vuser

The first step in creating a Java-compatible Vuser script is creating a Java Vuser template.

**To create a Java Vuser script:**

1. Open VuGen.
2. 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 lrapi.*;
public class Actions
{
	public int init() {
		return 0;
	}

	public int action() {
		return 0;
	}

	public int end() {
		return 0;
	}
}
```

The Actions class contains three methods: **init**, **action**, and **end**. The following table shows what to include in each method and when each method is executed.

| Script method | Used to emulate... | Is executed when... |
|---|---|---|
| *init* | a login to a server | the Vuser is initialized (loaded) |
| *action* | client activity | the Vuser is in "Running" status |
| *end* | a log off procedure | the Vuser finishes or is stopped |

## Init Method

Place all the login procedures and one-time configuration settings in the **init** method. The **init** method is only executed once—when the Vuser begins running the script. The following sample **init** method initializes an applet.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;

// Public function: init
    public int init() throws Throwable {

        // Initialize Orb instance...
        MApplet mapplet = new MApplet("http://chaos/classes/", null);
        orb = org.omg.CORBA.ORB.init(mapplet, null);
...
```

### Action Method

Place all Vuser actions in the **action** method. The **action** method is executed according to the number of iterations you set in the runtime settings. For more information on the iteration settings, see Chapter 12, "Configuring Run-Time Settings." The following sample **action** method retrieves and prints the Vuser ID.

```
public int action() {
     lr.message("vuser: " + lr.get_vuser_id() + " xxx");
          return 0;
 }
```

### End Method

In the **end** method, place the code you want the Vuser to execute at the end of the script, such as logging off from a server, cleaning up the environment, and so forth.

The **end** method is only executed once—when the Vuser finishes running the script. In the following example, the **end** method closes and prints the **end** message to the execution log.

```
public int end() {
      lr.message("End");
      return 0;
  }
```

# Java Vuser API Functions

VuGen provides a specific Java API for Java Vuser scripts. These functions are all static methods of the lrapi.lr class. For further information about each of these functions, refer to 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

| | |
|---|---|
| **lr.declare_transaction** | Declares a transaction. |
| **lr.start_transaction** | Marks the beginning of a transaction. |
| **lr.end_transaction** | Marks the end of a transaction. |

### Command Line Parsing Functions

| | |
|---|---|
| **lr.get_attrib_double** | Retrieves a **double** type variable used on the script command line. |
| **lr.get_attrib_long** | Retrieves a **long** type variable used on the script command line. |
| **lr.get_attrib_string** | Retrieves a string used on the script command line. |

### Informational Functions

| | |
|---|---|
| **lr.value_check** | Checks the value of a parameter. |
| **lr.user_data_point** | Records a user-defined data sample. |
| **lr.get_group_name** | Returns the name of the Vuser's group. |
| **lr.get_host_name** | Returns the name of the load generator executing the Vuser script. |
| **lr.get_master_host_name** | Returns the name of the machine running the LoadRunner Controller, or Administration Console. |
| **lr.get_object** | Captures a Java object and dumps it to a data file. (Corba-Java only) |

| | |
|---|---|
| **lr.get_scenario_id** | Returns the ID of the current scenario or session step. |
| **lr.get_vuser_id** | Returns the ID of the current Vuser. |

### String Functions

| | |
|---|---|
| **lr.deserialize** | Expands an object to represent its ASCII components. |
| **lr.eval_string** | Replaces a parameter with its current value. |
| **lr.eval_data** | Replaces a parameter with a byte value. |
| **lr.eval_int** | Replaces a parameter with an integer value. |
| **lr.eval_string** | Replaces a parameter with a string. |
| **lr.next_row** | Indicates to use the next row of data for the specified parameter. |
| **lr.save_data** | Saves a byte as a parameter. |
| **lr.save_int** | Saves an integer as a parameter. |
| **lr.save_string** | Saves a null-terminated string to a parameter. |

### Message Functions

| | |
|---|---|
| **lr.debug_message** | Sends a debug message to the Output window. |
| **lr.enable_redirection** | Enables the redirection of standard messages and errors to a log file, as standard output and standard error. |
| **lr.error_message** | Sends an error message to the Vuser log file and Output window with location details. |
| **lr.get_debug_message** | Retrieves the current message class. |
| **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. |

| | |
|---|---|
| **lr.redirect** | Redirects a string to a file. |
| **lr.set_debug_message** | Sets a debug message class. |
| **lr.vuser_status_message** | Sends a message to the Vuser Status area in the Controller/Console window. (LoadRunner only) |

**Run-Time Functions**

| | |
|---|---|
| **lr.declare_rendezvous** | Declares a rendezvous in a Vuser script. |
| **lr.peek_events** | Indicates where a Vuser script can be paused. |
| **lr.rendezvous** | Sets a rendezvous point in a Vuser script. |
| **lr.think_time** | 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 short or complex tasks. When working with LoadRunner, you can analyze the performance per transaction during and after the scenario run, using online monitor and graphs.

You can also specify a transaction status: lr.PASS or lr.FAIL. You can let the Vuser automatically determine if the transaction was successful, or you can incorporate it into a conditional loop. For example, in your code you can check for a specific return code. If the code is correct, you issue a lr.PASS status. If the code is wrong, you issue an lr.FAIL status.

**To mark a transaction:**

1. Insert **lr.start_transaction** into the script, at the point where you want to begin measuring the timing of a task.
2. Insert **lr.end_transaction** into the script, at the point where you want to stop measuring the task. Use the transaction name as it appears in the **lr.start_transaction** function.

**3** Specify the desired status for the transaction: lr.PASS or lr.FAIL.

```
public int action() {

for(int i=0;i<10;i++)
  {
     lr.message("action()"+i);
     lr.start_transaction("trans1");
     lr.think_time(2);
     lr.end_transaction("trans1",lr.PASS);
  }
     return 0;
}
```

## Inserting Rendezvous Points

The following section does not apply to Business Availability Center.

To emulate heavy user load on your client/server system, you synchronize Vusers to perform a task at exactly the same moment by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, it is held by the Controller until all Vusers participating in the rendezvous arrive.

You designate the meeting place by inserting a rendezvous function into your Vuser script.

**To insert a rendezvous point:**

➤ Insert an **lr.rendezvous** function into the script, at the point where you want the Vusers to perform a rendezvous.

```
public int action() {

  for(int i=0;i<10;i++)
  {
     lr.rendezvous("rendz1");
     lr.message("action()"+i);
     lr.think_time(2);
  }
  return 0;
}
```

## Obtaining Vuser Information

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

| | |
|---|---|
| **lr.get_attrib_string** | Returns a string containing command line argument values or runtime information such as the Vuser ID or the load generator name. |
| **lr.get_group_name** | Returns the name of the Vuser's group. |
| **lr.get_host_name** | Returns the name of the load generator executing the Vuser script. |
| **lr.get_master_host_name** | Returns the name of the machine running the LoadRunner Controller, Administration Console, or Tuning Module Console. |
| **lr.get_scenario_id** | Returns the ID of the current scenario or session step. (LoadRunner only) |
| **lr.get_vuser_id** | Returns the ID of the current Vuser. (LoadRunner only) |

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

```
String my_host = lr.get_host_name();
```

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

## Issuing Output Messages

When you run a scenario or session step, the Controller/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 Controller/Console. The Controller/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 433, or refer to the *Online Function Reference* (**Help > Function Reference**).

You can instruct the Vusers to redirect the Java standard output and standard error streams to VuGen's Execution log. This is especially helpful when you need to paste existing Java code or use ready-made classes containing **System.out** and **System.err** calls in your Vuser scripts. In the execution log, standard output messages are colored blue, while standard errors are shown in red.

The following example shows how to redirect specific messages to the standard output and standard error using **lr.enable_redirection:**

```
lr.enable_redirection(true);

System.out.println("This is an informatory message…");  // Redirected
System.err.println("This is an error message…");  // Redirected

lr.enable_redirection(false);

System.out.println("This is an informatory message…");  // Not redirected
System.err.println("This is an error message…");  // Not redirected
```

---

**Note:** When you set **lr.enable_redirection** to **true**, it overrides all previous redirections. To restore the former redirections, set this function to **false**.

---

For additional information about this function, 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 Vusers handle think time functions, open the runtime settings dialog box. For more information, see Chapter 12, "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 insert command line options after the script path and filename in the Controller, Tuning Module or Administration Console. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

| | |
|---|---|
| **lr.get_attrib_double** | Retrieves double precision floating point type arguments |
| **lr.get_attrib_long** | Retrieves long integer type arguments |
| **lr.get_attrib_string** | Retrieves character strings |

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

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

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

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

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

For more information on the command line parsing functions, refer to the *Online Function Reference* (**Help > Function Reference**). For more information on how to insert the command line options, refer to the *LoadRunner Controller*, *Tuning Module Console*, *Performance Center*, or *Business Availability Center* documentation.

## 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** subdirectory) and all of the Mercury classes (**classes** subdirectory).

➤ All classes used by the Java Vuser must be in the classpath—either set in the machine's CLASSPATH environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

## Running Java Vuser Scripts

Java Vuser scripts differ from C Vuser scripts in that they are first compiled and then executed; C Vuser scripts are interpreted. VuGen locates the **javac** compiler from within the JDK installation and compiles the Java code inside the script. This stage is indicated by the **Compiling...** status message in the bottom of the VuGen window. If errors occur during compilation, they are listed in the execution log. To go to the code in your script that caused the error, double-click on the error message containing the line number of the error. Fix the error and run the script again.

If the compilation succeeds, the status message **Compiling...** changes to **Running...** and VuGen begins to execute the script. When you run the script again, VuGen runs the script without recompiling it, provided that no changes were made to the script. To debug your script further, you can use breakpoints and animated run type execution using the step option.

---

**Note:** If you are making calls to JNDI extensions within your script, you may encounter problems trying to run your Vusers as **threads**. This happens because JNDI requires each thread to have its own context class loader. In order to run as threads, instruct each Vuser to run with its own context class loader, by adding the following line to the beginning of the **init** section:

```
DummyClassLoader.setContextClassLoader();
```

---

# Compiling and Running a Script as Part of a Package

When creating a Java Vuser script, you may need to use methods in other classes in which the class or method is protected. If you try to compile this type of script, you will receive errors in the compilation stage indicating that the methods are inaccessible. To make sure that your script can access these methods, insert the package name containing these methods at the top of the script, just as you would do in a standard Java program—<package_name>. In the following example, the script defines the **just.do.it** package which consists of a path:

```
package just.do.it;

import lrapi.*;
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 ready-made code segments into scripts or import ready-made classes in order to invoke their methods. If Vusers need to run as threads under the Controller/Console (for scalability reasons), you need to make sure that all of the imported code is thread-safe.

Thread-safety is often difficult to detect. A Java Vuser may run flawlessly under VuGen and under the Controller/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 lrapi.*;
public class Actions
{
  private static int iteration_counter = 0;

    public int init() {
        return 0;
    }

    public int action() {
        iteration_counter++;
        return 0;
    }

    public int end() {
        lr.message("Number of Vuser iterations: "+iteration_counter);
        return 0;
    }
}
```

When you run one Vuser, the **iteration_counter** member accurately determines the number of iterations that were executed. When multiple Vusers run together as threads on a single virtual machine, the static class member **iteration_counter** is shared by all threads, resulting in an incorrect counting. The total number of all Vusers iterations is counted.

If code is known to be non thread-safe and you still want to import it into your script, you can run the Vusers as processes. For more information on running Vusers as threads or processes, see Chapter 12, "Configuring Run-Time Settings."

When you run a basic Java Vuser script, it usually consists of a single thread—the main thread. Only the main thread can access the Java Vuser API. If a Java Vuser spawns secondary worker threads, using the Java API may cause unpredictable results. Therefore, it is recommended to use the Java Vuser API only in the main thread. Note that this limitation also affects the **lr.enable_redirection** function.

The following example illustrates where the LR API may and may not be used. The first log message in the execution log indicates that the value of **flag** is false. The virtual machine then spawns a new thread **set_thread**. This thread runs and sets **flag** to true, but will not issue a message to the log, even though the call to **lr.message** exists. The final log message indicates that the code inside the thread was executed and that **flag** was set to true.

```
boolean flag = false;

public int action() {
  lr.message("Flag value: "+flag);
  Thread set_thread = new Thread(new Runnable();{
          public void run() {
            lr.message("LR-API NOT working!");
            try {Thread.sleep(1000);} catch(Exception e) {}
            flag = true;
          }
  });
  set_thread.start();
  try {Thread.sleep(3000);} catch(Exception e) {}
  lr.message("Flag value: "+flag);
  return 0;
}
```

# Part VII

## Distributed Component Protocols

# 30

# 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 by using a Visual Basic add-in. For more information about the Visual Basic add-in, see Chapter 76, "Creating Vuser Scripts in Visual Studio."

Chapter 31, "Understanding COM Vuser Scripts," explains how VuGen COM scripts work and provides a brief function reference.

This chapter describes:

- About Recording COM Vuser Scripts
- 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 **lrc** prefix.

You can view and edit the recorded script from the VuGen's main window. The COM API/method calls that were recorded during the session are displayed in the window, allowing you to visually track application COM/DCOM calls.

You can indicate the programming language in which to create a Vuser script—either C or Visual Basic scripting. For more information, see Chapter 5, "Setting Script Generation Preferences."

## 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. Most users apply the Vusers to check the load on remote servers. Therefore, objects accessed by Remote Object Proxy are usually the most relevant for these tests.

### COM Data Types

COM also provides several special data types, including safe arrays, BSTR strings and variants. You may need to use these data types for debugging, parameterization and similar tasks.

## Getting Started with COM Vusers

This section describes the process of developing COM Vuser scripts.

**To develop a COM Vuser script:**

**1 Record the basic script using VuGen.**

Start VuGen and create a new Vuser script. Specify COM as the type of Vuser. Choose an application to record and set the recording options. To set the script related recording options, see Chapter 5, "Setting Script Generation Preferences." To set the COM specific options and filters, see the "Setting COM Recording Options" on page 456. Record typical operations using your application.

For details about recording, see Chapter 4, "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 7, "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 8, "Working with VuGen Parameters."

**5 Configure the run-time settings.**

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Selecting COM Objects to Record

The application you are testing may use a great many COM objects. Only a few may actually create load and may be important for the load test. Thus, before you record a COM application, you should select the objects you want to record for the load test. VuGen allows you to browse for objects from type libraries that it can read on the local machine and on other computers in the network.

### Deciding Which Objects to Use

There are several ways to decide which COM objects should be included in the test. Try to determine which remote objects are used by the software. If you are unsure which objects to choose, try using the default filter. The Environments branch of the filter includes calls to three sets of objects (ADO, RDS and Remote) that are likely to generate load on remote servers.

You can also check the actual calls to refine the filter. After you have recorded the test, you can save the file and look in the **data** directory that VuGen creates for a file named **lrc_debug_list_<nnn>.log**", where **nnn** is the process number. This log file contains a listing of each COM object that was called by the recorded application, regardless of whether or not the recording filter included that object. Only calls that generate load on the server should be included for recording.

For example, the following is a local COM of the Visual Basic library:

```
Class JetES {039EA4C0-E696-11D0-878A-00A0C91EC756}
was loaded from type library "JET Expression Service Type Library"
({2358C810-62BA-11D1-B3DB-00600832C573} ver 4.0)
```

It should not be added since it does not generate load on the server.

Likewise, since the OLE DB and Microsoft Windows Common Controls are local objects, the following are examples of classes and libraries that are not going to place any load on the server and should not be recorded:

```
Class DataLinks {2206CDB2-19C1-11D1-89E0-00C04FD7A829}
was loaded from type library "Microsoft OLE DB Service Component 1.0
Type Library"
({2206CEB0-19C1-11D1-89E0-00C04FD7A829} ver 1.0)

Class DataObject {2334D2B2-713E-11CF-8AE5-00AA00C00905}
was loaded from type library "Microsoft Windows Common Controls 6.0
(SP3)"
({831FDD16-0C5C-11D2-A9FC-0000F8754DA1} ver 2.0)
```

However, for example, a listing such as the following indicates a class that should be recorded since it does generate load on the server:

```
Class Order {B4CC7A90-1067-11D4-9939-00105ACECF9A}
was loaded from type library "FRS"
({B4CC7A8C-1067-11D4-9939-00105ACECF9A} ver 1.0)
```

Calls to classes of the **FRS** library, used for instance in the flight_sample that is installed with VuGen, use server capacity and should be recorded.

If a COM object itself calls other COM objects, all the calls will be listed in the type information log file. For example, every time the application calls an **FRS** class function, the **FRS** library calls the **ActiveX Data Object (ADO)** library. If several functions in such a chain are listed in a filter, VuGen records only the first call that initiates the chain. If you selected both **FRS** and **ADO** calls, only the **FRS** calls will be recorded.

On the other hand, if you select only the **ADO** library in the filter, then calls to the **ADO** library will be recorded. It is often easier to record the call to the first remote object in the chain. In some cases, however, an application may use methods from several different COM objects. If all of them use a single object that puts a load on the server, you could only record the final common object.

### Which Objects Can Be Selected

VuGen can only record objects if it can read their type libraries. If the type libraries were not installed in the system or VuGen cannot find them, the COM objects will not be listed in the Recording Options dialog box. If they are used by your application, VuGen will not be able to identify these objects and will identify them as **INoTypeInfo** in the files.

### Which Interfaces Can Be Excluded

For each object, the Recording Options dialog box will show you all interfaces that are listed in the Type Library, and allow you to specify inclusion or exclusion of each one. However, **ADO**, **RDS** and **Remote Objects** can be included in the filter as a group. The filter will not show the individual objects of those environments or their interfaces. Objects that you included from type libraries may also have interfaces that are not listed in the type library and therefore not shown in the Recording Options dialog. After generating a VuGen script, you can identify these interfaces in the script and get their GUID numbers from the interfaces.h file that VuGen generates. Using this information, you can exclude the interfaces as explained below.

# Setting COM Recording Options

Use the COM Recording Options dialog box to set the filtering and COM scripting options. You use the online browser to locate type libraries in the registry, file system, or the Microsoft Transaction Server (MTS).

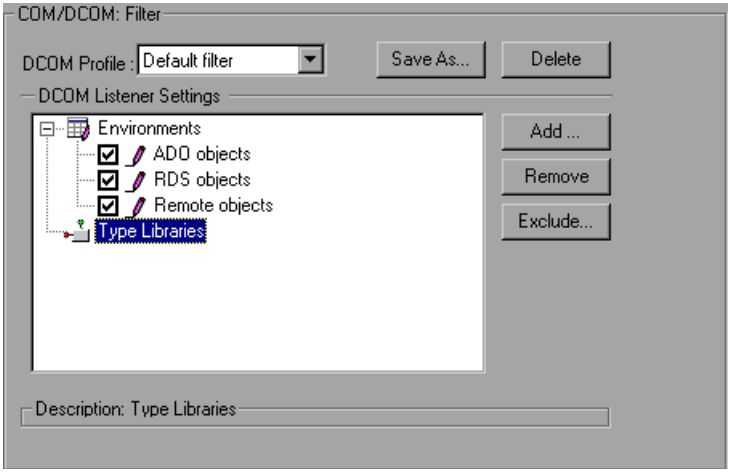
For more information, see:

- Filtering Objects
- Setting the Filter
- Setting COM Scripting Options

## Filtering Objects

The Filter options let you indicate which COM objects should be recorded by VuGen. You can select objects from within environments and libraries.

The Filter options set a default filter or create alternate filters. You can filter a recording session by environment and type libraries.



### DCOM Profile

- **Default Filter:** The filter to be used as the default when recording a COM Vuser script.
- **New Filter:** A clean filter based on the default environment settings. Note that you must specify a name for this filter before you can record with its settings.

### DCOM Listener Settings

The DCOM Listener Settings display a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, a dialog box opens asking you if you also want to exclude the interface in all classes that implement it this interface.

Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

- **Environment:** The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record.
- **Type Libraries:** A type library **.tlb** or **.dll** file, that represents the COM object to record. All COM objects have a type library that represents them. You can choose a type library from the Registry, Microsoft Transaction Server, or file system.

**Type Libraries:** In the lower section of the dialog box, VuGen displays the following information for each type library.

- **TypLib:** The name of the type library (tlb file).
- **Path:** The path of the type library.
- **Guid:** The Global Unique Identifier of the type library.

### Setting the Filter

This section describes how to set the filters.

**To select which COM objects to record:**

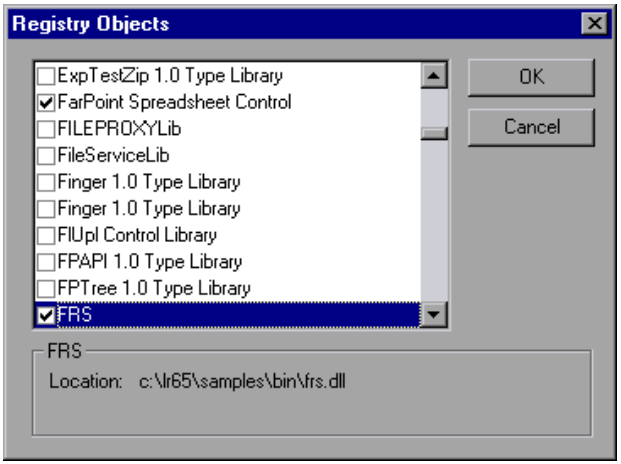
1 Choose **Tools** > **Recording Options** from the main menu or click **Options** in the Start Recording dialog box. A dialog box opens displaying the Recording Options tree. Select the **COM/DCOM:Filter** node.

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. Clear the check box adjacent to **ADO**, **RDS** or **Remote objects** to exclude them from the filter.

2 Click **Add** to add another COM type library, and select a source to browse: registry, file system, or MTS, as described below.

**3** Select **Browse Registry** to display a list of type libraries found in the registry of the local computer.




Select the check box next to the desired library or libraries and click **OK**.

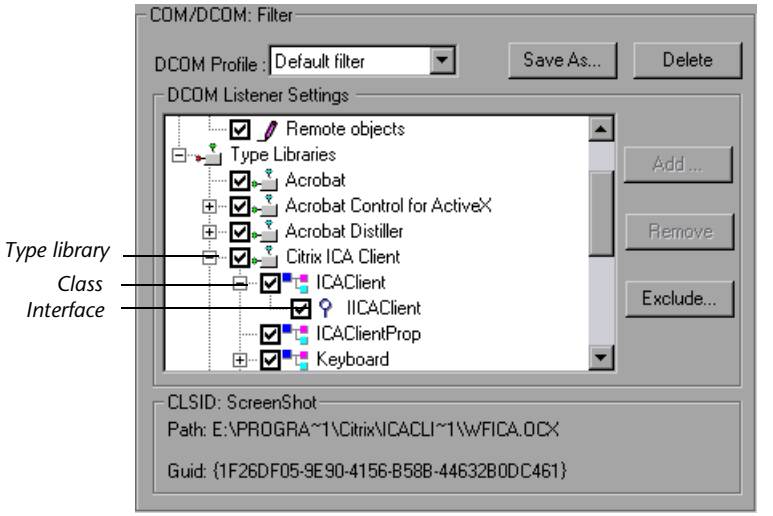
**4** To add a type library from the file system, click **Add** and select **Browse file system**.

Select the desired file and click **OK**.

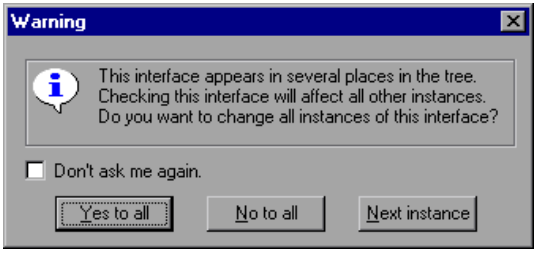
**5** Once the type library appears in the list of Type Libraries, you can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class.

To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item.

Note that when you clear a check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box.

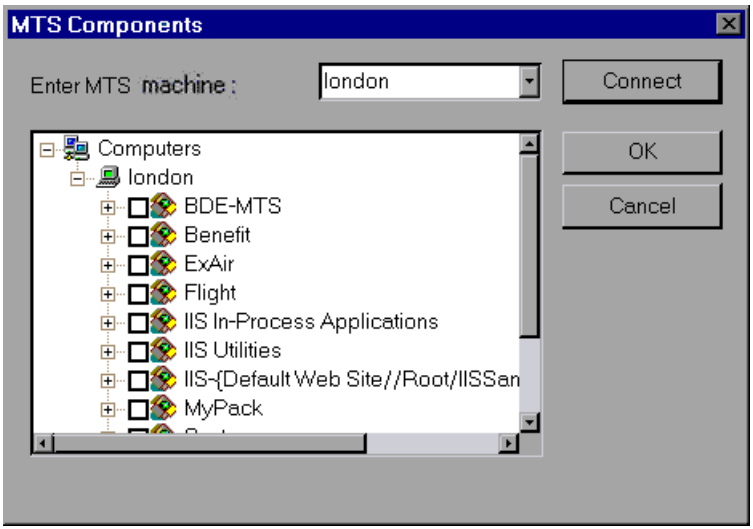



**6** An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the following warning:




If you check **Don't ask me again** and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click **Yes to all** to change the status of all instances of this interface for all other classes, click **No to all** to leave the status of all other instances unchanged. Click **Next Instance** to view the next class that uses this interface.

**7** To add a component from a Microsoft Transaction Server, click **Add** and select **Browse MTS**. The MTS Components dialog box prompts you to enter the name of the MTS server.

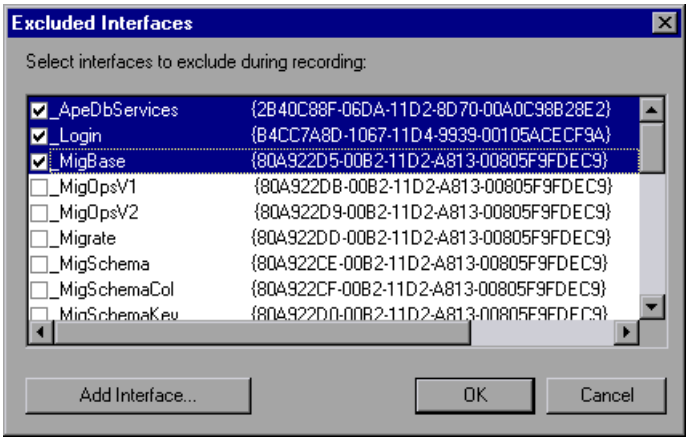



Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine.

Select one or more packages of MTS components from the list of available packages and click **Add**. Once the package appears in the list of Type Libraries, you can select specific components from the package.

**8** In addition to disabling and enabling recording of interfaces in the tree display, you can also click **Exclude** in the Recording Options dialog to include or exclude interfaces in the filter, whatever their origin.

Note that you can also exclude classes and interfaces by clearing the check box adjacent to the item, inside the type library tree hierarchy.



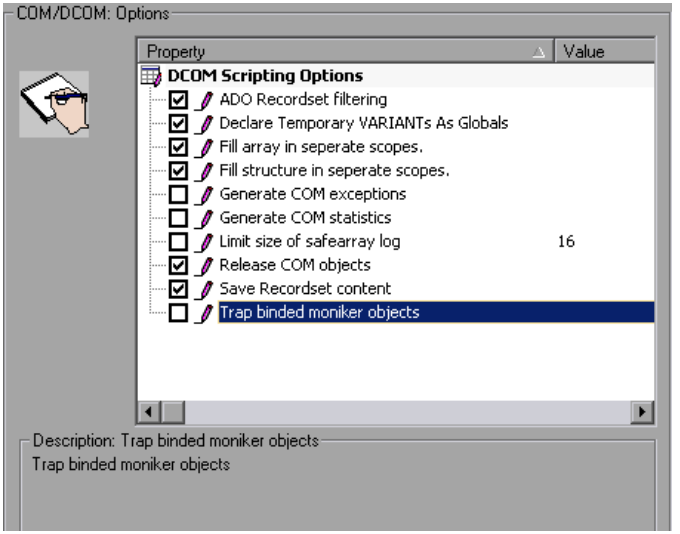

The checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click **Add Interface...** in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the **Add Interface…** feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter.

**9** When you have finished modifying your filter, click **OK** to save it and close the dialog box. Click **Save As** to save a **New filter**, or to save an existing filter under a new name. You can select saved filters in subsequent recordings. Default settings are given initially in the **Default filter**.

## Setting COM Scripting Options

You can set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.




**ADO Recordset filtering:** Condense multiple recordset operations into a single-line fetch statement (enabled by default).

**Declare Temporary VARIANTs as Globals:** Define temporary VARIANT types as Globals, not as local variables (enabled by default).

**Fill array in separate scopes:** Fill in each array in a separate scope (enabled by default).

**Fill structure in separate scopes:** Fill in each structure in a separate scope (enabled by default).

**Generate COM exceptions:** Generate COM functions and methods that raised exceptions during recording (disabled by default).

**Generate COM statistics:** Generate recording time performance statistics and summary information (disabled by default).

**Limit size of SafeArray log:** Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).

**Release COM Objects:** Record the releasing of COM objects when they are no longer in use (enabled by default).

**Save Recordset content:** Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).

**Trap binded moniker objects:** Trap all of the bound moniker objects (disabled by default).

**To set COM/DCOM options:**

1. 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 the check boxes adjacent to them.
3. Click **OK** to save your settings and exit.

# 31

# 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, see Chapter 30, "Recording COM Vuser Scripts."

This chapter describes:

- About COM Vuser Scripts
- Understanding VuGen COM Script Structure
- Examining Sample VuGen COM Scripts
- Scanning a Script for Correlations
- Correlating a Known Value

***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.

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 interface header file defines the interface pointers, as well as other variables, that can be used in the script. Each interface has an Interface ID (IID) which uniquely identifies the interface.

The format of the interface definition is:

**<interface type>*<interface name> = 0; //"{<IID of the interface type>}"**

In the following example, the interface type is **IDispatch**, the name of the interface instance is **IDispatch_0**, and the **IID** of **IDispatch** type is the long number string:

```
IDispatch* IDispatch_0= 0;//"{00020400-0000-0000-C000-
000000000046}"
```

### Vuser Script Statements

The COM Vuser script consist of code that creates object instances, retrieves interface pointers and calls the interface methods. Each user action may generate one or more COM calls. Each COM call is coded by VuGen as a group of statements. Each such group is contained in a separate scope enclosed in braces. Several different statements prepare for the main call by assigning values and performing type conversions. For example, the group of calls needed to create an object may look like this:

```
{
GUID pClsid = lrc_GUID("student.student.1");
IUnknown * pUnkOuter = (IUnknown*)NULL;
unsigned long dwClsContext = lrc_ulong("7");
GUID riid = IID_IUnknown;
lrc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid,
(void**)&IUnknown_0, CHECK_HRES);
}
```

### Error Checking

Each COM method or API call returns an error value. VuGen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

| | |
|---|---|
| **CHECK_HRES** | This value is inserted if the function passed during recording and errors should be checked during replay. |
| **DONT_CHECK_HRES** | This value is inserted if the function failed during recording and errors should not be checked during replay. |

# Examining Sample VuGen COM Scripts

This section shows examples of how VuGen emulates a COM client application.

## Basic COM Script Operations

The basic operations are:

- Instantiation of the object
- Retrieving interface pointers
- Calling interface methods

Each type of operation is done within a separate scope.

## Instantiation of the Object

To use a COM object, the application must first instantiate it and get a pointer to an interface of that object.

**VuGen does the following to instantiate an object:**

1. VuGen calls **lrc_GUID** to get a unique ProgID for the object, to be stored in **pClsid**:

```
GUID pClsid = lrc_GUID("student.student.1");
```

**pClsid** is the unique global CLSID of the object, which was converted from the ProgID **student.student.1**

2. If the unknown interface pointer is a pointer to an aggregated object, VuGen retrieves the pointer to that object, or else it sets it to NULL:

```
IUnknown * pUnkOuter = (IUnknown*)NULL;
```

**3** VuGen sets the contexts of the object to be created:

```
unsigned long dwClsContext = lrc_ulong("7");
```

**dwClsContext** contains the context of the object (in process, local, remote or combinations of these.)

**4** VuGen sets a variable to hold the requested interface ID, which is **IUnknown** in this case:

```
GUID riid = IID_IUnknown;
```

**riid** contains the interface ID of the **IUnknown** interface.

**5** After the input parameters are prepared, a call to **lrc_CoCreateInstance** creates an object using the parameters defined in the preceding statements. A pointer to the **IUnknown** interface is assigned to output parameter IUnknown_0. This pointer is needed for subsequent calls:

```
lrc_CoCreateInstance(&pClsid, pUnkOuter, dwClsContext, &riid,
(void**)&IUnknown_0, CHECK_HRES);
```

**The input parameters were prepared and explained above.** Since the call succeeded, VuGen sets error checking on during the user simulation by inserting the **CHECK_HRES** value. **The call returns a pointer to the IUnknown interface in IUnknown_0,** that can be used in subsequent calls.

### Retrieving an Interface

After creating an object, VuGen has access only to the **IUnknown** interface**.** VuGen will use the **IUnknown** interface for communicating with the object. This is done using the **QueryInterface** method of the **IUnknown** standard interface. The first parameter in a VuGen method call is the interface instance. In this case it is the **IUnknown_0** pointer set previously by **CoCreateInstance**. The **QueryInterface** call requires as input the ID of the interface to be retrieved, and returns a pointer to the interface designated by that ID.

**To get the interface:**

1 First, VuGen sets a parameter, **riid**, equal to the ID of the Istudent interface:

```
GUID riid = IID_Istudent;
```

2 A call to **QueryInterface** assigns a pointer to the **Istudent** interface to output parameter Istudent_0 if the Istudent object has such an interface:

```
lrc_IUnknown_QueryInterface(IUnknown_0, &riid, (void**)&Istudent_0,
CHECK_HRES);
```

### Using an Interface to Set Data

Here is an example of using the methods of the interface to set data. Suppose that in the application, the user is supposed to input a name. This activates a method for setting the name. VuGen records this in two statements. One statement is used for setting up the name string and the second one sets the name property.

**To set up the entire function call:**

1 First, VuGen sets a variable (**Prop Value**) equal to the string. The parameter is of type BSTR, a string type used in COM files:

```
BSTR PropValue = lrc_BSTR("John Smith");
```

In subsequent stages, you will probably parameterize this call, replacing "John Smith" with a parameter, so that different names are used each time the Vuser script is run.

2 Next, VuGen calls the **Put_Name** method of the **Istudent** interface to enter the name:

```
lrc_Istudent_put_name(Istudent_0, PropValue, CHECK_HRES);
```

### Using an Interface to Return Data

Returning data from an application is different than entering the data, because you might want to store these values and use them as inputs in subsequent calls for parameterization.

**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:

```
lrc_Istudent_get_name(Istudent_0, &pVal, CHECK_HRES);
```

**3** VuGen then generates a statement for saving the values:

```
//lrc_save_BSTR("param-name",pVal);
```

The statement is commented out. You can remove the comments and change <param-name> to a variable with a meaningful name to be used for storing this value. VuGen will use the variable to save the value of **pVal** returned by the previous call. You can then use the variable as a parameterized input in subsequent calls to other methods.

## The IDispatch Interface

Most COM objects have specific interfaces. Many of them also implement a general-purpose interface called **IDispatch**, which VuGen translates in a special way. IDispatch is a "superinterface" that exposes all of the other interfaces and methods of a COM object. Calls to the **IDispatch:Invoke** method from VuGen scripts are implemented using **lrc_Disp** functions. These calls are constructed somewhat differently from calls to other interfaces.

The **IDispatch** interface **Invoke** method can execute a method, it can get a property value, or it can set a value or reference value for a property. In the standard **IDispatch:Invoke** method these different uses are signalled in a **wflags** parameter. In the VuGen implementation they are implemented in different procedure calls that invoke a method or put or get a property.

For example, a call to IDispatch to activate the **GetAgentsArray** method may look like this:

```
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray",
/*locale*/1033, LAST_ARG, CHECK_HRES);
```

The parameters in the above call are:

| | |
|---|---|
| **IDispatch_0** | This is the pointer to the **IDispatch** interface returned by a previous call to the **IUnknown:Queryinterface** method. |
| **GetAgentsArray** | This is the name of the method to invoke. Behind the scenes, VuGen will get the ID of the method from the name. |
| **1033** | This is the language locale. |
| **LAST_ARG** | This is a flag to tell the **IDispatch** interface that there are no more arguments. |
| **CHECK_HRES** | This flag turns on checking of HRES, since the call succeeded when it was recorded. |

In addition, there might be another parameter, **OPTIONAL_ARGS**. This signals that in addition to any standard parameters, VuGen is sending some optional arguments. Each optional argument consists of a pair giving the ID or name of the argument and its value. For example, the following call to lrc_DispMethod passes optional arguments "#3" and "var3":

```
{
        GUID riid = IID_IDispatch;
        lrc_IOptional_QueryInterface(IOptional_0, &riid,
(void**)&IOptional_0, CHECK_HRES);
    }
    {
        VARIANT P1 = lrc_variant_short("47");
        VARIANT P2 = lrc_variant_short("37");
        VARIANT P3 = lrc_variant_date("3/19/1901");
        VARIANT var3 = lrc_variant_scode("4");
        lrc_DispMethod((IDispatch*)IOptional_0, "in_out_optional_args",
/*locale*/1024, &P1, &P2, OPTIONAL_ARGS, "#3", &P3, "var3", &var3,
LAST_ARG, CHECK_HRES);
```

The different **lrc_Disp** methods that use the **IDispatch** interface are detailed in Chapter 32, "Understanding COM Vuser Functions".

### 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 Chapter 32, "Understanding COM Vuser Functions." Previously, we showed how the **lrc_DispMethod1** call was used to retrieve an array of name strings:

```
VARIANT retValue = lrc_variant_empty();
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray",
/*locale*/1033, LAST_ARG, CHECK_HRES);
```

The following example now shows how VuGen gets the strings out of **retValue**, which is a variant that will be read as an array of strings.

First, VuGen extracts the BSTR array from the variant:

```
BstrArray array0 = 0;
array0 = lrc_GetBstrArrayFromVariant(&retValue);
```

With all the values in **array0**, VuGen provides you with code that you can use to extract the elements from the array for later use in parameterization, as in the example below:

```
//GetElementFrom1DBstrArray(array0, 0); // value: Alex
//GetElementFrom1DBstrArray(array0, 1); // value: Amanda
....
```

VuGen has numerous type conversion functions and functions for extracting conventional types from variants. These are detailed in Chapter 32, "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 and assist you in getting a successful replay. It performs the following steps:

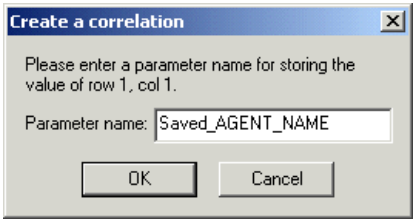
- scans for potential correlations
- insert the appropriate correlation function to save the results to a parameter
- replace the statement value with the parameter

You can perform automatic correlation on the entire script, or at a specific location in your script.

This section describes how to determine the statement which needs to be correlated. If you already know which value you want to correlate, proceed to the next section for instructions on correlating a specific value.

**To scan and correlate a script detected with automatic correlation:**

1. Select **View** > **Output** to display the output tabs at the bottom of the window. Check for errors in the Replay Log tab. Often, these errors can be corrected by correlation.
2. Select **Vuser** > **Scan for Correlations**. VuGen scans the entire script and lists all possible values to correlate in the **Correlated Query** tab.
3. Correlate the value. In the Correlated Query tab, double-click on the result you want to correlate. This is located on the 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.
4. In the grid, choose **Create Correlation** from the right-click menu. VuGen prompts you to enter a parameter name for the result value.




5. Specify a name, or accept the default. Click **OK** to continue. VuGen inserts the appropriate correlation statement (**lrc_save_**<type>) which saves the result to a parameter.
6. Click **Yes** to confirm the correlation.
7. A message box opens asking if you want to search for all occurrences of the value in the script.
   Click **No** to replace only the value in the selected statement.
   To search and replace additional occurrences click **Yes**.
8. A Search and Replace dialog box opens. Confirm any replacements, including your original statement.
9. Close the Search and Replace dialog box. VuGen replaces the statement value with a reference to the parameter. Note that if you choose to cancel the correlation, VuGen also erases the statement created in the previous step.

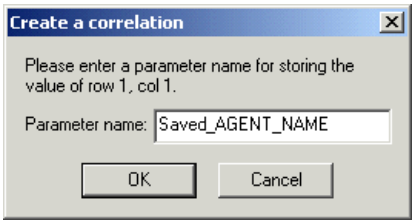
## Correlating a Known Value

If you know which value needs to be correlated, perform the following procedure:

**To correlate a specific value:**

1. Locate the argument you want to correlate (usually in an **lrc_variant_** statement) 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.
3. 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.
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 (**lrc_save**_<type>) which saves the result to a parameter.

```
lrc_save_rs_param (Recordset20_0, 1, 1, 0, “Saved_AGENT_NAME”);
```

6. Click **Yes** to confirm the correlation.
7. A message box opens asking if you want to search for all occurrences of the value in the script.

Click **No** to replace only the value in the selected statement.
To search and replace additional occurrences click **Yes**.

**8** A Search and Replace dialog box opens. Confirm any replacements, including your original statement.

# 32

# Understanding COM Vuser Functions

The COM Vuser functions emulate the actions of a user running a COM application.

This chapter describes:

- About COM Vuser Functions
- 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 lrc prefix. VuGen records the COM API and method calls listed in this section. You can also manually program lrc type conversion calls. For syntax and examples of the lrc 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 5, "Setting Script Generation Preferences." 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:

| | |
|---|---|
| **lrc_CoCreateInstance** | Creates an instance of an object and returns the unknown interface. |
| **lrc_CreateInstanceEx** | Creates an instance of an object on a remote machine and can return multiple interfaces. |
| **lrc_CoGetClassObject** | Fetches the class factory for the specified class. The class factory can then be used to create multiple objects of that class. |
| **lrc_Release_Object** | 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**:

| | |
|---|---|
| **lrc_DispMethod** | Invokes a method of an interface using the **IDispatch:Invoke** method. |
| **lrc_DispMethod1** | Invokes a method and gets a property of the same name using the IDispatch interface. |
| **lrc_DispPropertyGet** | Gets a property using the IDispatch interface. |
| **lrc_DispPropertyPut** | Sets a property using the IDispatch interface. |
| **lrc_DispPropertyPutRef** | 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:

**lrc_<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:

**lrc_save_<Type-Name>**

**lrc_save_VARIANT_<Type-Name>**

Saves a variable of the given <Type-Name> as a variant.

**lrc_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:

| | |
|---|---|
| **lrc_CoObject_from_variant** | Extracts a pointer to an IUnknown interface from a variant. |
| **lrc_CoObject_by_ref_from_variant** | Extracts a pointer to an IUnknown interface from a reference within a variant. |
| **lrc_DispObject_from_variant** | Extracts a pointer to an IDispatch interface from a variant. |
| **lrc_DispObject_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:

| | |
|---|---|
| **lrc_variant_<Type-Name>Array** | Assigns an array of type **<Type-Name>** to a variant. |
| **lrc_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. |
| **lrc_Get<Type-Name>ArrayFromVariant** | Extracts an array of Type-Name from a variant. |
| **lrc_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 **lrc_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 **lrc_CreateVBCollection**. Thereafter, it can refer to the data at the interface address.

## ADO RecordSet Functions

The following are ADO recordset functions

| | |
|---|---|
| **lrc_FetchRecordset** | Moves a pointer through a recordset. |
| **lrc_FetchRecordsetUntillEOF** | Fetches records until the end of the recordset. |
| **lrc_RecordsetWrite** | Updates a field in an ADO recordset. |
| **lrc_RecordsetAddColumn** | Adds a new column to a recordset. |
| **lrc_RecordsetDeleteColumn** | Deletes a column from a recordset. |

## Debug Functions

The **lrc_print_variant** function prints the contents of a variant.

## VB Collection Support

The **lrc_CreateVBCollection** function creates a Visual Basic Collection object.

# 33

# 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 Vuser API Java-specific functions.

This chapter describes:

- About Developing Corba-Java Vuser Scripts
- Recording a Corba-Java Vuser
- Working with Corba-Java Vuser Scripts
- Recording on Windows XP and Windows 2000 Servers
- Application Specific Tips

***The following information applies to Corba-Java Vuser scripts.***

## About Developing 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 Vuser API 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.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

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. See Chapter 29, "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 Servers" on page 498.

The next few chapters discuss the Java recording options, run-time settings, and correlation.

## Recording a Corba-Java Vuser

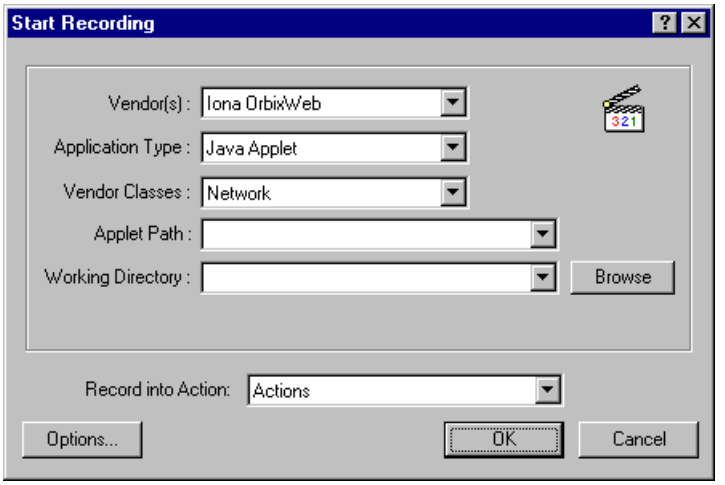
Before recording a Corba-Java Vuser, verify that your application or applet functions properly on the recording machine.

Ensure that you have properly installed a JDK version from Sun on the machine running VuGen—JRE alone is insufficient. You must complete this installation before recording a script. Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions.

For more information on the required environment settings, see Chapter 29, "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 **IExplore** 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 necessary only 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, see Chapter 18, "Setting Java Recording Options."

**7** In the **Record into Action** box, select the section corresponding to the method into which you want to record. The Actions class contains three methods: **init**, **action**, and **end**, corresponding to the vuser_init, Actions, and vuser_end sections. The following table shows what to include into each method, and when each method is executed.

| method within Actions class | Record into action | Used to emulate... | Executed during... |
|---|---|---|---|
| **init** | **vuser_init** | a login to a server | Initialization |
| **action** | **Actions** | client activity | Running |
| **end** | **vuser_end** | a log off procedure | Finish or Stopped |

**Note:** Make sure to import the **org.omg.CORBA.ORB** function in the **vuser_init** section, so that it will not be repeated for each iteration.

**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.lr;

public class Actions {

  // Public function: init
  public int init() throws Throwable {

  // Initialize Orb instance...
  MApplet mapplet = new MApplet("http://chaos/classes/", null);
  orb = org.omg.CORBA.ORB.init(mapplet, null);

  // Bind to server...
  grid = grid_dsi.gridHelper.bind("gridDSI", "chaos");
  return lr.PASS;
}
```

The org.omg.CORBA.ORB function makes the connection to ORB. Therefore, it should only be called once. When running multiple iterations, place this function in the **init** section.

In the following section, VuGen recorded the actions performed upon a grid CORBA object.

```
// Public function: action
public int action() throws Throwable {

   grid.width();
   grid.height();
   grid.set(2, 4, 10);
   grid.get(2, 4);

   return lr.PASS;
}
```

At the end of the session, VuGen recorded the shutdown of the ORB. The variables used 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;
}
```

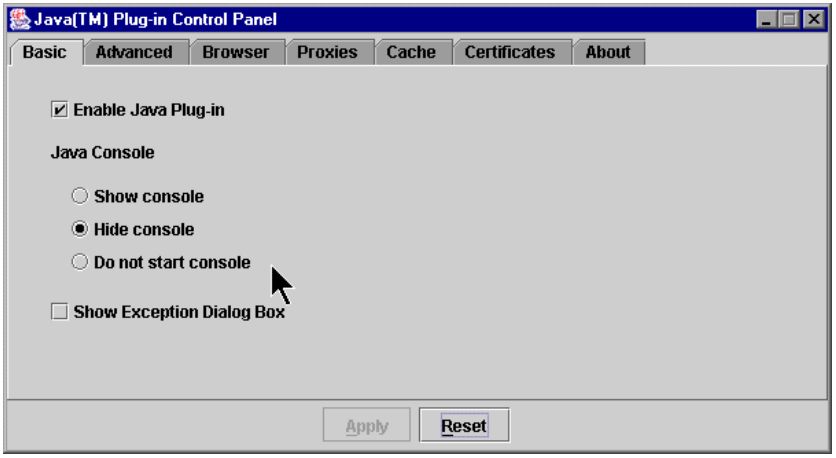
Note that the ORB shutdown statement was customized for this product. This customization prevents a single Vuser's shutdown from shutting down all other Vusers.

# Recording on Windows XP and Windows 2000 Servers

When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

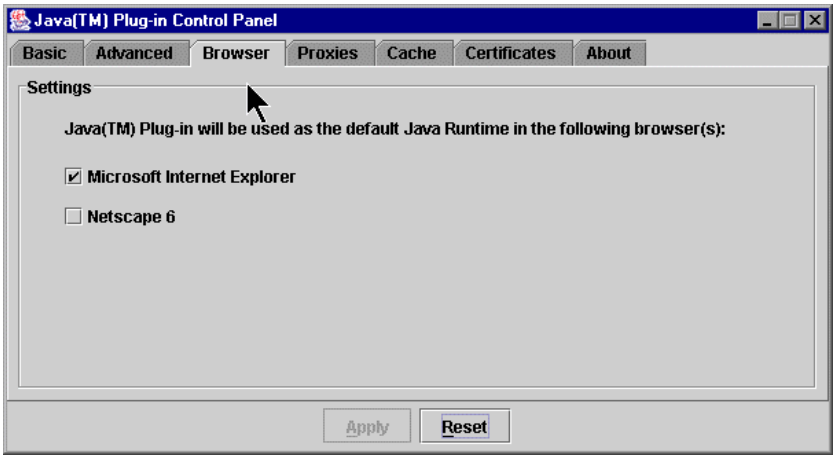
**To configure your machine for a Corba-Java or Rmi-Java recording:**

**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.**

## Application Specific Tips

Running CORBA applications with JDK1.2 or later, might load the JDK internal CORBA classes instead of the specific vendor CORBA classes. To force the virtual machine to use the vendor classes, specify the following java.exe command-line parameters:

**Visigenic 3.4**

```
-Dorg.omg.CORBA.ORBClass=com.visigenic.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.visigenic.vbroker.orb.
    ORBSingleton
```

**Visigenic 4.0**

```
-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton
```

**OrbixWeb 3.x**

```
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.
    singletonORB
```

**OrbixWeb 2000**

```
-Dorg.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.
    ORBSingleton
```

# 34

# 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 Vuser API Java-specific functions.

This chapter describes:

- About Developing RMI-Java Vuser Scripts
- 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 Vuser API 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.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. See Chapter 29, "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 Servers" on page 498.

## 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

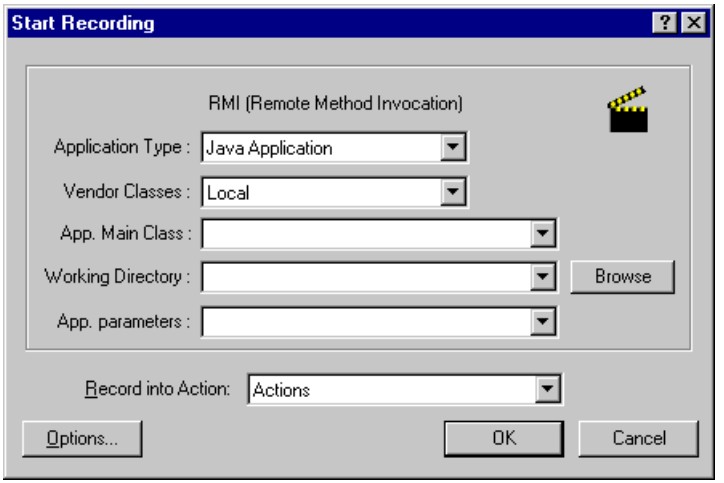
Before recording an RMI Vuser, verify that your application or applet functions properly on the recording machine.

Ensure that you have properly installed a JDK version from Sun on the machine running the script—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 29, "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 VuGen.

**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 **IExplore** 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, see Chapter 18, "Setting Java Recording Options."

**6** In the **Record into Action** box, select the section corresponding to the method into which you want to record. The Actions class contains three methods: **init**, **action**, and **end**, corresponding to the vuser_init, Actions, and vuser_end sections. The following table shows what to include into each method, and when each method is executed.

| method within Actions class | Record into action | Used to emulate... | Executed during... |
|---|---|---|---|
| **init** | **vuser_init** | a login to a server | Initialization |
| **action** | **Actions** | client activity | Running |
| **end** | **vuser_end** | a log off procedure | Finish or Stopped |

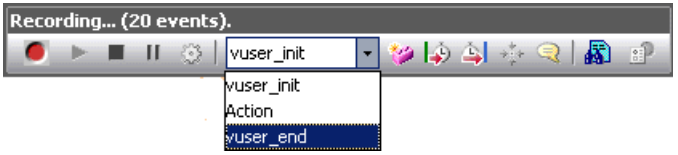
**7** Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.




**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** section from the floating toolbar.




Perform the log off procedure. VuGen records the procedure into the **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 a lookup operation to obtain a specific Java object. You 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 290.

# Part VIII

## E-Business Protocols

# 35

# Developing FTP Vuser Scripts

VuGen allows you to emulate network activity by directly accessing an FTP server.

This chapter describes:

- About Developing FTP Vuser Scripts
- 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 4, "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 4, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Working with FTP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 5, "Setting Script Generation Preferences." 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, refer to 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;
}
```

# 36

# Developing LDAP Vuser Scripts

VuGen allows you to emulate the communication with an LDAP server.

This chapter describes:

- About Developing LDAP Vuser Scripts
- 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 "Defining Distinguished Name Entries" on page 519.

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 4, "Recording with VuGen."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Working with LDAP Functions

You can indicate the programming language in which to create a Vuser script. For more information, see Chapter 5, "Setting Script Generation Preferences." 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, refer to the *Online Function Reference* (**Help > Function Reference**).

In the following example, the user logs on to an LDAP server, ldap1. It adds an entry and then renames the OU attribute from **Sales** to **Marketing**.

```
Action()
{
        // Logon to the LDAP server
        mldap_logon("Login",
                "URL=ldap://johnsmith:tiger@ldap1:80",
                LAST);

        // Add an entry for Sally R. Jones
        mldap_add("LDAP Add",
                "DN=cn=Sally R. Jones,OU=Sales, DC=com",
                "Name=givenName", "Value=Sally", ENDITEM,
                "Name=initials", "Value=R", ENDITEM,
                "Name=sn", "Value=Jones", ENDITEM,
                "Name=objectClass", "Value=contact", ENDITEM,
                LAST);

        // Rename Sally's OU to Marketing
        mldap_rename("LDAP Rename",
                "DN=CN=Sally R. Jones,OU=Sales,DC=com",
                "NewDN=OU=Marketing",
                LAST);

        // Logout from the LDAP server
        mldap_logoff();
    return 0;
}
```

# Defining Distinguished Name Entries

The LDAP API references objects by its **distinguished name** (DN). A DN is a sequence of **relative distinguished names** (RDN) separated by commas.

An RDN is an attribute with an associated value in the form attribute=value. The attribute names are not case-sensitive. The following table lists the most common RDN attribute types.

| String | Attribute Type |
|---|---|
| **DC** | domainComponent |
| **CN** | commonName |
| **OU** | organizationalUnitName |
| **O** | organizationName |
| **STREET** | streetAddress |
| **L** | localityName |
| **ST** | stateOrProvinceName |
| **C** | countryName |
| **UID** | userid |

The following are examples of distinguished names:

```
DN=CN=John Smith,OU=Accounting,DC=Fabrikam,DC=COM
DN=CN=Tracy White,CN=admin,DC=corp,DC=Fabrikam,DC=COM
```

The following table lists reserved characters that cannot be used in an attribute value.

| Character | Description |
|---|---|
| | space or # character at the beginning of a string |
| | space character at the end of a string |
| , | comma |
| + | plus sign |
| " | double quote |
| \ | backslash |
| < | left angle bracket |
| > | right angle bracket |
| ; | semicolon |

To use a reserved character as part of an attribute value, you must precede it with an escape character, a backslash (\). If an attribute value contains other reserved characters, such as the equal sign (=) or non-UTF-8 characters, you must encode it in hexadecimal format—a backslash followed by two hex digits.

The following are examples of DNs that include escaped characters. The first example is an organizational unit name with an embedded comma; the second example is a value containing a carriage return.

```
DN=CN=Bitwise,OU=Docs\, Support,DC=Fabrikam,DC=COM
DN=CN=Before\0DAfter,OU=Test,DC=North America,DC=Fabrikam,DC=COM
```

# 37

# Recording Microsoft .NET Vuser Scripts

VuGen records applications that were created in the .NET Framework environment.

This chapter describes:

- About Recording Microsoft .NET Vuser Scripts
- Getting Started with Microsoft .NET Vusers
- Setting Microsoft .NET Recording Options
- Viewing Scripts in VuGen and Visual Studio
- Adding .NET References in the Run-Time Settings
- Viewing Data Sets and Grids
- Troubleshooting Your Script
- Correlating Microsoft .NET Scripts

***The following information only applies to Microsoft .NET Vuser scripts.***

# About Recording Microsoft .NET Vuser Scripts

Microsoft's .NET Framework provides a a solid foundation for developers to build various types of applications such as ASP.NET, Windows Forms, Web Services, distributed applications, or applications that combine several of these models.

VuGen allows you to create Vuser scripts that emulate users of Microsoft .NET client applications created in its .NET Framework. VuGen records all of the clients actions through methods and classes, and creates a script in VB .NET.

By default, the VuGen environment is configured for ADO .NET applications. Contact Customer Support for information on how to configure VuGen to record applications created with other models.

## Limitations

The following limitations apply to the VuGen recording of a Microsoft .NET application:

- Recording is only supported for .NET Framework 1.1.
- It is recommended that you install .NET Framework 1.1 before installing VuGen. If you install .NET Framework after VuGen, you must manually install the MICGenericHook by running installmicgenerichook.msi in the **<LoadRunner root>\bin** folder.
- Events are not supported.
- Microsoft .NET scripts only support single-protocol recording in VuGen.
- Direct access to public fields are not supported—you must access fields through methods.
- VuGen does not record static fields in the applications—it only records methods within classes.
- VuGen is only able to detect classes in a DLL file—it is unable to detect them in an exe file.
- Multi-threaded support is dependent on the client application. If the recorded application supports multi-threading, then the Vuser script will also support multi-threading.

# Getting Started with Microsoft .NET Vusers

This section describes the process of developing Microsoft .NET Vuser scripts.

**To develop a .NET Vuser script:**

**1 Record the basic script using VuGen.**

Start VuGen and create a new Vuser script. Specify **Microsoft .NET** as the type of Vuser. Choose an application to record and set the recording options. To set the script related recording options, see Chapter 5, "Setting Script Generation Preferences."

For details about recording, see Chapter 4, "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 7, "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 about parameterization properties, see Chapter 8, "Working with VuGen Parameters."

**4 Configure the run-time settings.**

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see "Viewing Scripts in VuGen and Visual Studio" on page 525

**5 Run the script from VuGen.**

Save and run the script from VuGen to verify that it runs correctly.

For details, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

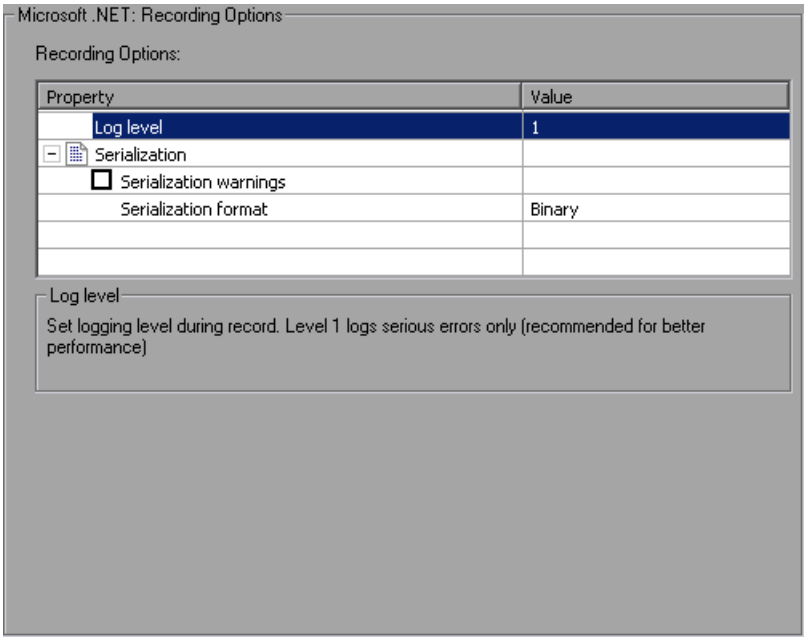
After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Setting Microsoft .NET Recording Options

You can set both General and Microsoft .NET-specific recording options. This section describes the .Microsoft .NET recording options. For information on the Script recording options, see Chapter 5, "Setting Script Generation Preferences."

To open the .NET Recording Options dialog box, choose **Tools** > **Recording Options**. You can set one of the following options:

- Log Level
- Serialization Settings



### Log Level

The Log Level options let you set the level of detail in the recording log file: To disable logging, set the level to 0. Level 1 is the basic level of logging. It creates a code generation log and logs all errors related to the instrumentation process.

### Serialization Settings

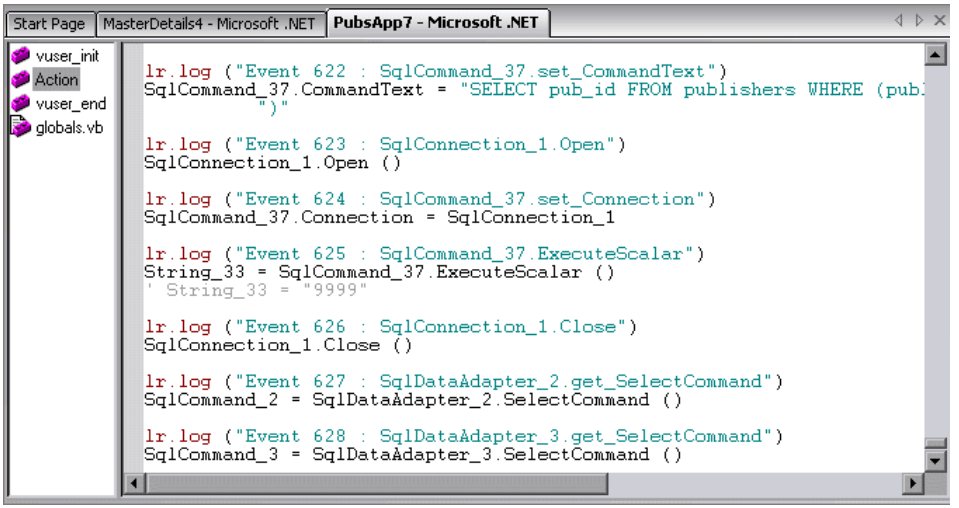
The Serialization options allow you to indicate whether or not to issue serialization warnings and let you set the serialization format.

**Serialization warnings:** Issue serialization warnings during recording.

**Serialization format:** The format of the serialization file that VuGen creates while recording a class that supports serialization: bin (binary), XML, or both.

## Viewing Scripts in VuGen and Visual Studio

After the recording, you can view the script in VuGen Script view. VB functions represent all of the actions you performed in your application during the recording session.




VuGen records the events that occurred, wrapping them in **lr.log** calls.

For methods that support serialization, VuGen casts the serialized objects as shown in the following example:

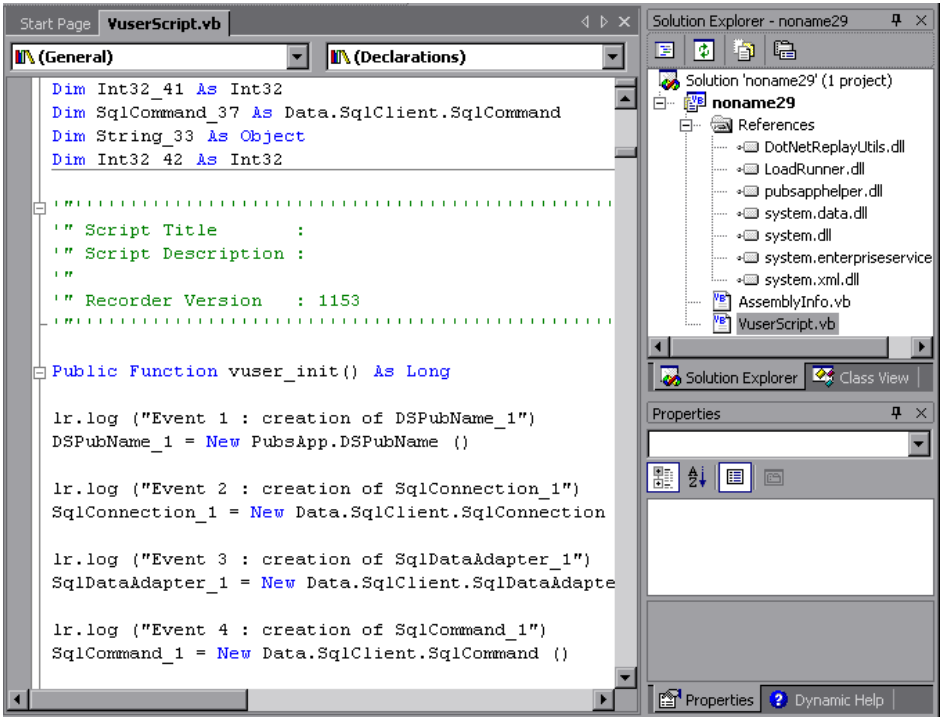
```
CultureInfo_1 = DirectCast(Utils.GetSerializedObject("CultureInfo_1.bin"),
                    System.Globalization.CultureInfo)
lr.log ("Event 20 : DSPubName_1.set_Locale")
DSPubName_1.Locale = CultureInfo_1
```

To run the script directly from VuGen, press F5 or choose **Vuser** > **Run**.

When you replay the script, first VuGen compiles it to ensure that all of the calls are valid and that the syntax is correct. VuGen compiles the VB calls into a DLL file with the name <script_name>.dll, which it saves in the script's **dat** folder. This single DLL file contains three functions - Init, Actions, and End.

VuGen also creates a Visual Studio solution file (sln extension) in your script's **Solution** folder.

You can open the file in Visual Studio .NET and view all of its components in the Solution Explorer. Click on **VuserScript.vb** to view the contents of the script.




To debug the script and run it with breakpoints or step-by-step, run it from within Visual Studio .NET—breakpoints and step-by-step replay are not supported in VuGen's editor window for Microsoft .NET Vusers.

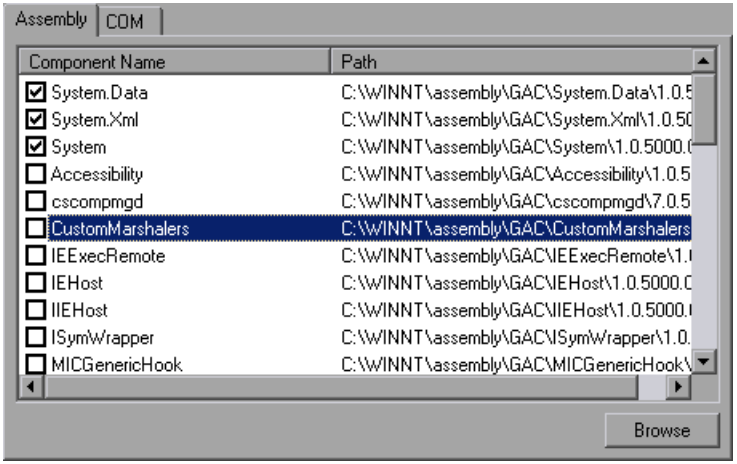
Note that VuGen automatically loads all of the necessary references that were required during recording. To add additional references for replay, use the Solution Explorer or VuGen's Run-Time settings. For more information, see "Viewing Scripts in VuGen and Visual Studio" on page 525.

**Note:** Before opening a script in Visual Studio .NET, make sure that you compile the script after saving it. If you resave the script to a new location, you must recompile it before opening it in Visual Studio.

## Adding .NET References in the Run-Time Settings

Before running your Microsoft .NET Vuser script, you can specify the .NET references to use during compilation from the Run-Time Settings dialog box.

Note that in a normal recording, VuGen automatically includes the references that are required for the replay of the script. You can find these references in the list of Component Name selected with check marks.

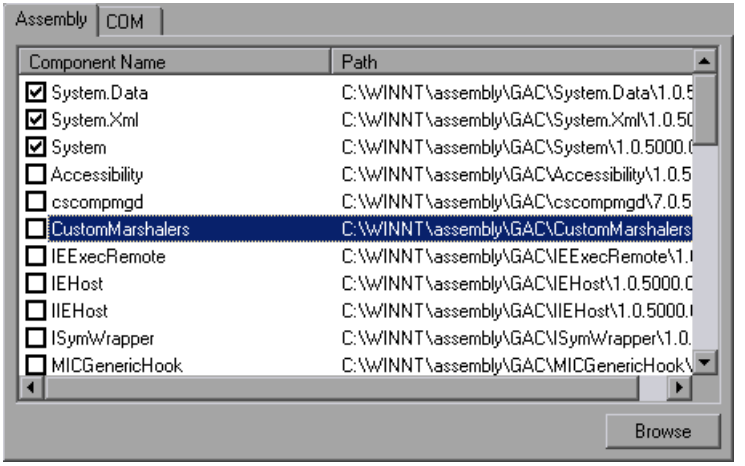



You can specify additional references in both Assembly and COM, by selecting the reference in the **Component Name** list.

**To specify .NET resources:**

1. Open the run-time setting—press F4 or choose **Vuser** > **Run-Time Settings**.
2. Click on the **.NET References node**. in the left pane.

**3** Click the **Assembly** tab. The screen shows all the Assembly libraries in the Gac—those that were automatically added to the script and those added manually by the user.




**4** To add additional Assembly files, click **Browse** and locate the file.

**5** To use the resource, select the checkbox adjacent to the resource.

**6** Click the **COM** tab. VuGen shows all of the registered COM classes.

**7** To use the resource, select the checkbox adjacent to the resource.

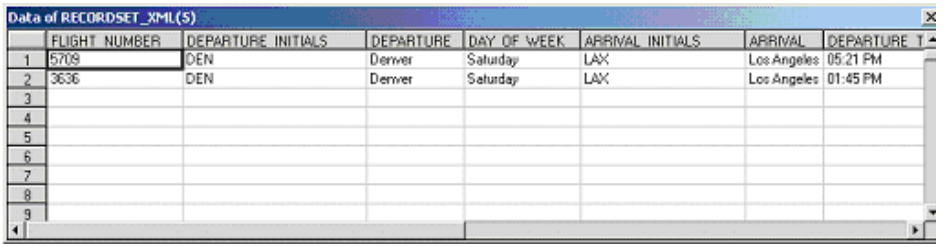
**8** Click **OK** to store the settings and close the dialog box.

You can also set general run-time settings for your Microsoft .NET script for configuring the pacing and iteration options. For more information, see Chapter 12, “Configuring Run-Time Settings.”

In the Speed Emulation run-time settings, you set the network speed that you want to emulate. For more information, see Chapter 13, “Configuring Network Run-Time Settings.”

# Viewing Data Sets and Grids

When you record a data set, VuGen generates a grid for the data.




To close the grid, click on the "-" in the margin adjacent to the beginning of the grid. The VuGen editor hides the grid and displays:

```
DATASET_XML(20)
```

The data set is stored in an XML file. You can view this XML file in the script's data/datasets folder, using the index number in the script as an indicator to locate the file. For example, DATASET_XML(20) would be represented by 20.xml.

By default, VuGen displays the grids in your script. To disable the grid display and instruct VuGen to show the collapsed version of the grid, select **View** > **Enable Data Grids** to remove the check mark. For more information, see "Grids" on page 364.

# Troubleshooting Your Script

The following section provides tips for successfully recording a .NET application with VuGen.

### Security Exceptions

A Security Exception that occurs while recording an application is usually due to a lack of permissions—the recording machine does not have sufficient permissions to record the application. This is common where your application is not local, but on the Intranet or network.

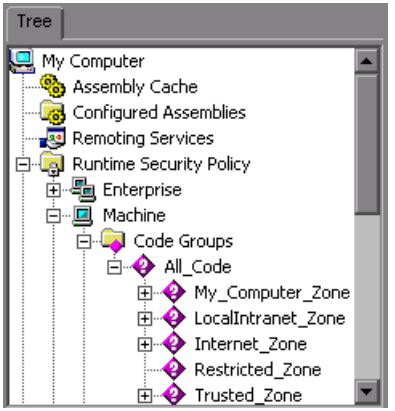
To solve this problem, you need to allow the recording machine to access the application and the script with Full Trust.

One solution is to copy the application and save your script locally, since by default, Vusers have Full Trust permissions to all local applications and folders.

An additional solution is to create new code groups that gives Full Trust to each application folder, and the script folder.

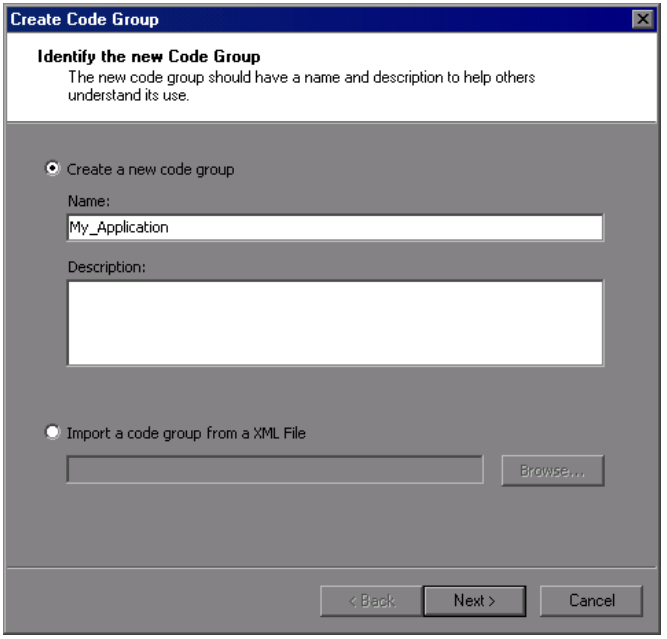
**To grant Full Trust permissions to a specific folder:**

1. Open the .NET Configuration settings. Choose **Start** > **Programs** > **Administrative Tools** > **Microsoft .NET Framework 1.1 Configuration**. The .NET Configuration window opens.
2. Expand the **Runtime Security Policy** node to show the Code Groups of the machine.



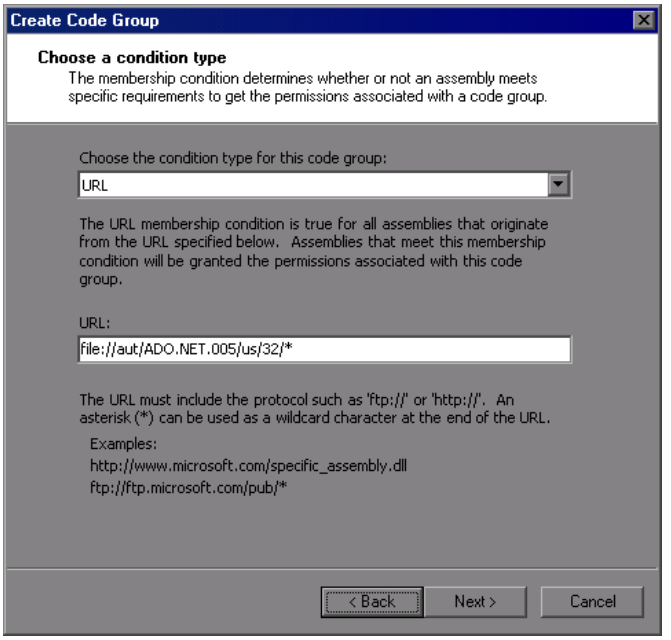
**3** Select the **All_Code** node.

**4** Choose **Action** > **New** …. The Create New Code Group dialog box opens.

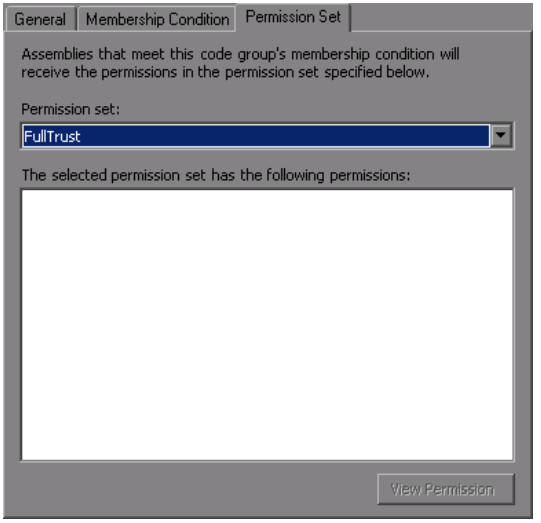



**5** Enter a name for a new Code Group for your application or script. Click **Next**.

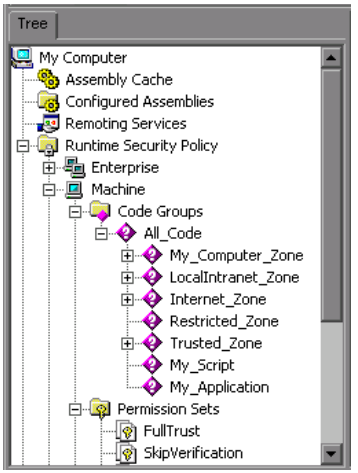
**6** Select the **URL** condition type. In the URL box, specify the full path of the application or script in the format file://... and click **Next**.




**7** Choose the **FullTrust** permission set. Click **Next**.



**8** Click **Finish** in the Completing the Wizard dialog box. The configuration tool adds your Code Group to the list of existing groups.




**9** Repeat the above procedure for all .NET applications that you plan to record.

**10** Repeat the above procedure for the Vuser script folder.

---

**Note:** Make sure that the script folder has **FullTrust** permissions on all Load Generator machines that are participating in the test (LoadRunner only).

---

# Correlating Microsoft .NET Scripts

After you record a session, you may need to **correlate** one or more values within your script. Correlating a value means that you capture a value during the script replay, and save it to a parameter. You can then use this parameter at a later point in the script.

To correlate a value within your Microsoft .NET Vuser script, you manually insert a correlation function, **lr.save_string**, in order to save the desired value to a parameter.

We will use the following script segment to illustrate correlation. In the following segment, the **SqlDataAdapter_1Fill** function retrieves the **CustomersAndOrdersDataSet_2** DataSet from the server which contains the **Customers** table/

```
lr.log ("Event 11 : SqlDataAdapter_1.Fill")
Int32_1 = SqlDataAdapter_1.Fill (CustomersAndOrdersDataSet_2, "Customers")

'CustomersAndOrdersDataSet_2.Tables("Customers") :
```

| | CustomerID | CompanyName | ContactName | ContactTitle | Address |
|---|---|---|---|---|---|
| 1 | ABC | ABC Company | John Smith | Owner | One My Way |
| 2 | ALFKI | Alfreds Futterkiste | Maria Anders | Sales Representative | Obere Str. 57 |
| 3 | ANATR | Ana Trujillo Emparec | Ana Trujillo | Owner | Avda. de la C |
| 4 | ANTON | Antonio Moreno Tac | Antonio Moreno | Owner | Mataderos 2 |
| 5 | AROUT | Around the Horn | Thomas Hardy | Sales Representative | 120 Hanover |

In the next section, we create a statement that saves the first CustomerID to a string inside the **MyCustomerID** variable (defined in globals.vb). This ID is located in the first row, **Row (0)**, and first column, **CustomerID**, of the table.

```
MyCustomerID = CustomersAndOrdersDataSet_2.Tables("Customers").Rows(0)(CustomersAndOrdersDataSet_2.Tables("Customers").Columns("CustomerID") ).ToString()
```

Next, we save the string to a Vuser parameter using **lr.save_string**. This prepares the string for the Vuser and allows it to be called during replay.

```
' Save it as LR parameter
lr.save_string( "MyCustomerID", CustomerID_param )
```

Finally, we use the parameter by calling it at a later point in the script.

```
' Use LR parameter for correlation
lr.log( lr.eval_string( "{CustomerID_param}" ) )
```

For more information about using correlation functions, refer to the *Online Function Reference*.

**To correlate a value**

1 Declare a variable in the **globals.vb** section.

```
Dim MyCustomerID As string
```

2 Locate the dataset in your script. After the dataset is returned, save the desired value from the dataset to the defined variable. For example:

```
DATASET_XML(0)

' Get the first CustomerID value returned from the DB
MyCustomerID = CustomersAndOrdersDataSet_2.Tables("Custom-
ers").Rows(0)(CustomersAndOrdersDataSet_2.Tables("Customers").Col-
umns("CustomerID")).ToString()
```

3 Insert the **lr.save_string** function and define a parameter. For example:

```
lr.save_string("MyCustomerID", CustomerID_param)
```

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

```
lr.log( lr.eval_string("{CustomerID_param}"))
```

# 38

# 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:

- About Developing Web Vuser Scripts
- 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) 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.

After you create a Vuser script, you run the script in standalone mode using VuGen. When the execution is successful, you are ready to integrate the Vuser script into a scenario or session step. For details on how to integrate a Vuser script into a scenario or session step, refer to the *LoadRunner Controller User's Guide* and 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 (for example, 20 seconds)—even when a large number of users (for example 200) access the site simultaneously. You use Vusers to emulate this case, where the Web server services simultaneous requests for information. Each Vuser could do the following:

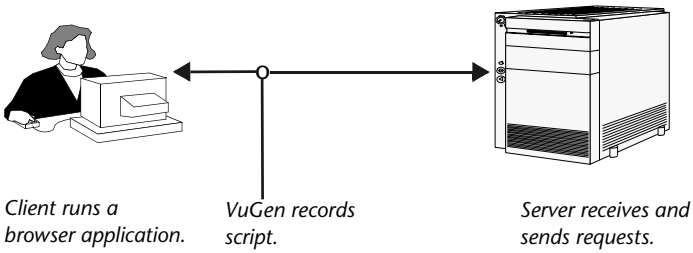
- Load a home page
- Navigate to the page containing the product information
- Submit a query
- Wait for a response from the server

You can distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many users.

The program that contains the calls to the server API is called a Vuser script. It emulates a browser application and all of the actions performed by the browser. Using the Console or Controller, you assign the script to multiple Vusers. The Vusers execute the script and emulate user load on the Web server.

# Understanding Web Vuser Technology

VuGen creates Web Vuser scripts by recording the activity between a browser and a Web server. VuGen monitors the client (browser) end of the system and traces all the requests sent to, and received from, the server.




When you run a recorded Vuser script, either in VuGen or from the Tuning Module 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 4, “Recording with VuGen.”

**2 Set the recording options.**

Set the recording options. For information about setting common Internet recording options, see Chapter 40, “Setting Recording Options for Internet Protocols.”

For details about Web specific recording options, see Chapter 41, “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 4, “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 44, “Verifying Web Pages Under Load”, Chapter 45, “Modifying Web and Wireless Vuser Scripts”, and Chapter 46, “Setting 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 8, “Working with VuGen Parameters.”

**6 Configure the run-time settings.**

The run-time settings control Vuser behavior during script execution. These settings include general run-time settings (iteration, log, think time, and general information), and Web-related settings (proxy, network, and HTTP details).

For details, see Chapter 12, "Configuring Run-Time Settings."

**7 Perform correlation.**

Scan your Vuser script for correlations and use one of VuGen's mechanisms to implement them.

For details, see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts" and Chapter 47, "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 14, "Running Vuser Scripts in Standalone Mode" and Chapter 49, "Using Reports to Debug Vuser Scripts."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Recording a Web Session

When you record a Web session, VuGen monitors all the actions that you perform in your Web browser. Your activities can include hyperlink jumps (both hypertext and hypergraphic) and form submissions. While recording, VuGen saves the recorded actions in a Web Vuser script.

Each Vuser script that you create contains at least three sections: **vuser_init**, one or more **Actions**, and **vuser_end**. During recording, you can select the section of the script into which VuGen will insert the recorded functions. The vuser_init and vuser_end sections are generally used for recording server 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.

VuGen creates a script describing user actions. By default, it generates a script with 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 it resembles a context sensitive recording.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("MercuryWebTours",
        "URL=http://localhost/MercuryWebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

# Converting Web Vuser Scripts into Java

VuGen provides a utility that enables you to convert a script created for a Web Vuser into a script for Java Vusers. This also allows you to create a hybrid Vuser script for both Web and Java.

**To convert a Web Vuser script into a Java Vuser script:**

1. Create an empty Java Vuser script and save it.
2. Create an empty Web Vuser script and save it.
3. Record a web session using standard HTML/HTTP recording.
4. Replay the Web Vuser script. When it replays correctly, cut and paste the entire script into a text document and save it as a text **.txt** file. In the text file modify any parameter braces from the Web type, "{ }" to the Java type, "< >".
5. Open a DOS command window and go to your product's **dat** directory.
6. Type the following command:

   ```
   <application_directory>\bin\sed -f web_to_java.sed filename > outputfilename
   ```

   where **filename** is the full path and filename of the text file you saved earlier and **outputfilename** is the full path and filename of the output file.
7. Open the output file, and copy its contents into your Java Vuser script action section at the desired location. If you are pasting the contents into an empty custom Java template (Java Vuser type), modify the line containing public int action() as follows:

   ```
   public int action() throws Throwable
   ```

   This change is done automatically for recorded Java users (RMI and CORBA).

   Parameterize and correlate the Vuser script as you would with an ordinary Java script and run it.

# 39

# 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:

- About Web Vuser Functions
- Adding and Editing Functions
- Web Function List
- Improving Performance Using Caching

***The following information applies to Web and Wireless 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, 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).
- As a text-based representation of the Vuser script. This is known as the **Script view**.

For more information, see “Viewing and Modifying Vuser Scripts” on page 17.

## Adding and Editing Functions

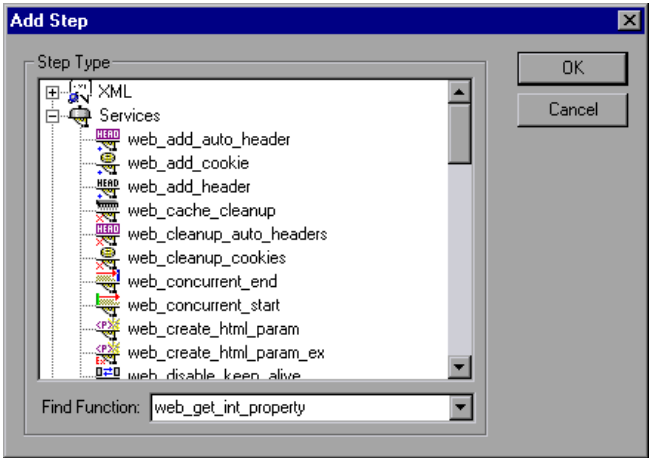
Many of the Web Vuser functions are recorded during the browser or toolkit session.

You can manually add general Vuser functions such as transactions, rendezvous, comments, and log functions during recording. For more information, see Chapter 7, “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. This dialog box lets you specify the function's arguments.




**3** Specify the properties and click **OK**. VuGen inserts the function with its arguments at the location of the cursor.

You can edit existing steps by opening the Properties dialog box and modifying the argument values. This is only valid for protocols that support tree view (not available for WAP).

**To edit an existing step:**

**1** In the tree view, select **Properties** from the right-click menu. The Properties dialog box for that function opens.

**2** Modify the argument values as necessary and click **OK**.

## 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

GUI-level Vusers use additional functions to emulate user actions. For more information, see "Using GUI-Level Vuser Functions" on page 832.

## 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. |
| **web_dump_cache** | Dumps the resources into the browser cache. |
| **web_load_cache** | Loads the contents of the cache. |

## 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

| | |
|---|---|
|  **web_create_html_param** | Saves dynamic information on an HTML page to a parameter (LR 6.5 and earlier). |
|  **web_create_html_param_ex** | Creates a parameter based on the dynamic information contained in an HTML page - uses embedded boundaries. (LR 6.5 and below) |
|  **web_reg_save_param** | Creates a parameter based on the dynamic information contained in an HTML page - does not use embedded boundaries. |
|  **web_set_max_html_param_len** | Sets the maximum length of retrieved dynamic HTML information. |

## Filter Functions

| | | |
|---|---|---|
|  | **web_add_filter** | Sets criteria to includes or exclude URLs when downloading. |
|  | **web_add_auto_filter** | Sets criteria to includes or exclude URLs when downloading. |
|  | **web_remove_auto_filter** | Disables filtering of download content. |

## Header Functions

| | | |
|---|---|---|
|  | **web_add_auto_header** | Adds a customized header to all subsequent HTTP requests. |
|  | **web_add_header** | Adds a customized header to the next HTTP request. |
|  | **web_cleanup_auto_headers** | 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

| | |
|---|---|
|  **web_convert_param** | Converts an HTML parameter to a URL or plain text. |
|  **web_get_int_property** | Returns specific information about the previous HTTP request. |
|  **web_report_data_point** | Specifies a data point and adds it to test results. |
|  **web_set_option** | Sets a Web option in the area of encoding, redirection, and downloading of non-HTML resources. |
|  **web_set_sockets_option** | 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:



**lr_start_transaction** Marks the beginning of a transaction for performance analysis or tuning.



**lr_end_transaction** Marks the end of a transaction for performance analysis or tuning.



**lr_rendezvous** Sets a rendezvous point in the Vuser script.



**lr_think_time** Pauses execution between commands in a Vuser script.

For more information on adding general Vuser functions to scripts, see Chapter 7, "Enhancing Vuser Scripts."

The following step types are supported in VuGen:

| Step Type | Description |
|---|---|
| Service | A **Service** step is a function that does not make any changes in the Web application context. Rather, service steps perform customization tasks such as setting proxies, providing authorization information, and issuing customized headers. |
| URL | A **URL** 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 (only recorded for the HTML-based recording level). |

| Step Type | Description |
|---|---|
| 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 (only recorded for the HTML-based recording level). |
| Submit Form / Submit Data | VuGen adds a **Submit Form** or **Submit Data** step when you submit a form while recording. The default label of the step is the name of the executable program used to process the form (Submit Form only recorded for the HTML-based recording level). |
| Custom Request | VuGen adds a **Custom Request** step to a Vuser script when you record an action that VuGen 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. |

# Improving Performance Using Caching

By utilizing the cache-simulating capabilities of VuGen, you can substantially improve user performance. The caching option reduces the CPU usage by approximately 15%.

To implement caching within your script, you manually add the **web_dump_cache** and **web_load_cache** functions.

### Dumping Information to the Cache

The first step in implementing caching, is dumping the information to a cache file. You run the **web_dump_cache** function to create a cache file in the location specified in the **FileName** argument. Note that you only need to run this function once to generate the cache file.

In the following example, the **web_dump_cache** function creates a cache file in **C:\temp** for each **VuserName** parameter running the script.

```
web_dump_cache("paycheckcache","FileName=c:\\temp\\{VuserName}paychec
k", "Replace=yes", LAST)
```

If you run a single Vuser user ten times, VuGen creates ten cache files in the following format, where the prefix is the VuserName value:

```
Ku001paycheck.cache
Ku002paycheck.cache
Ku003paycheck.cache
…
```

You can modify the first and second arguments (in this example paycheckcache and paycheck) to reflect the current transaction name. Place this function at the end of your script, after you have loaded all of the resources.

### Loading Information from the Cache

The final step in implementing caching, is loading the information stored in the cache file. The **web_load_cache** function loads a cache file whose location is specified in the **FileName** argument. Note that the **web_load_cache** function requires the cache file to exist. Therefore, you can only run this function after running **web_dump_cache**.

In the following example, the **web_load_cache** function loads the **paycheck** cache files from **C:\temp**.

```
web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",LA
ST)
```

### Inserting the Caching Functions

To implement caching in your script, you must first store the information to a cache file. During replay, each Vuser calls this information.

**To use the caching functions:**

1. Insert the **web_dump_cache** function at the beginning of your script.
2. Run the script at least once.

3 Insert the **web_load_cache** function into your script, before the Vuser actions.

4 Comment out the **web_dump_cache** function.

5 Run and save the script.

### Caching Example

The following example illustrates a PeopleSoft Enterprise Vuser viewing the details of his paycheck.

```
Action()
{
//     web_add_cookie("storedCookieCheck=true; domain=pbntas05;
path=/");

web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}pay-
check",LAST);

      web_browser("signon.html",
           DESCRIPTION,
           ACTION,
           "Navigate=http://pbntas05:8200/ps/signon.html",
           LAST);
      lr_think_time(35);

      web_edit_field("userid",
           "Snapshot=t1.inf",
           DESCRIPTION,
           "Type=text",
           "Name=userid",
           ACTION,
           "SetValue={VuserName}",
           LAST);
```

```
        web_edit_field("pwd",
            "Snapshot=t2.inf",
            DESCRIPTION,
            "Type=password",
            "Name=pwd",
            ACTION,
            "SetValue=HCRUSA_KU0007",
            LAST);

        lr_start_transaction("login");
            web_button("Sign In",
                    "Snapshot=t3.inf",
                    DESCRIPTION,
                    "Type=submit",
                    "Tag=INPUT",
                    "Value=Sign In",
                    LAST);
        lr_end_transaction("login", LR_AUTO);

        web_image_link("CO_EMPLOYEE_SELF_SERVICE",
            "Snapshot=t4.inf",
            DESCRIPTION,
            "Alt=",
            "Name=CO_EMPLOYEE_SELF_SERVICE",
            "Ordinal=1",
            LAST); …

        web_text_link("Sign out",
            "Snapshot=t7.inf",
            DESCRIPTION,
            "Text=Sign out",
            "FrameName=UniversalHeader",
            LAST);
    web_dump_cache("paycheck","FileName=c:\\{VuserName}paycheck",
    "Replace=yes", LAST);
        return 0;
    }
```

# 40

# Setting Recording Options for Internet Protocols

For protocols that work over the Internet, you can customize the Internet related recording options.

This chapter describes:

- About Setting Recording Options for Internet Protocols
- 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. 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.**

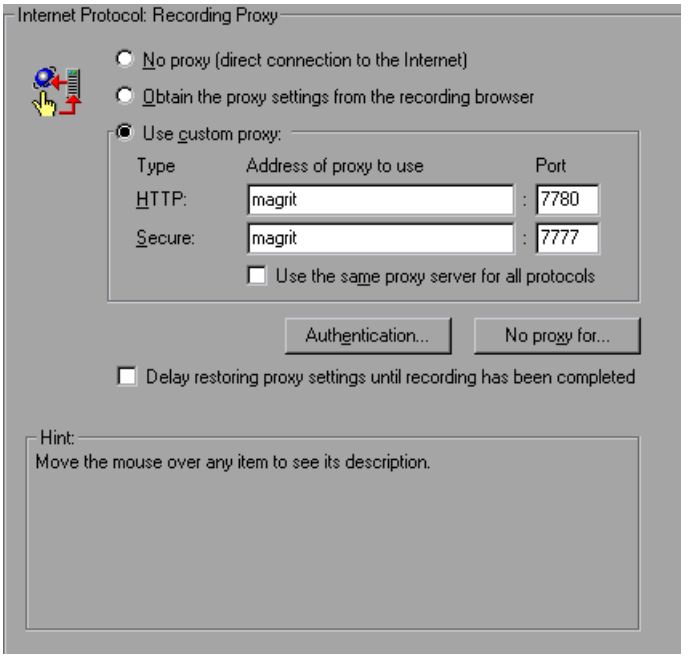
## Working with Proxy Settings

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.



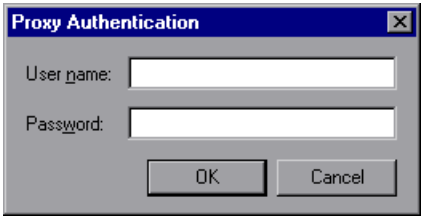

You can choose one of the following proxy options:

- **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.

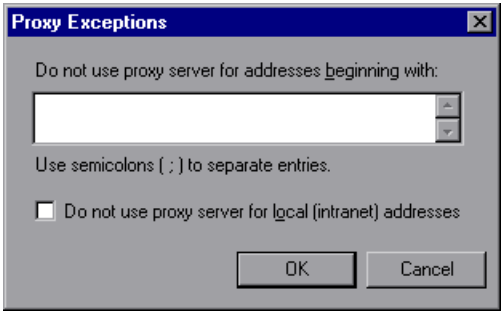
➤ **Use custom proxy.** Use the specified proxy server during recording. You can specify a proxy server for all non-secure HTTP sites and another proxy server for all secure (HTTPS) sites. This section is only enabled when the two above options are cleared.

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 set the recording options in the following areas:

- Internet Preferences Recording Options
- Selecting a Recording Engine
- Setting a Recording Scheme

## Internet Preferences Recording Options

The Internet Preference options allow the customization of code generation settings in the area of think time, resetting contexts, saving snapshots, and the generation of **web_reg_find** functions. Note that some of these options are not available in multi-protocol mode.

**Reset context for each action:** (Web, Oracle NCA only) This setting, enabled by default, tells VuGen to reset all HTTP contexts between actions. Resetting contexts allows the Vuser to more accurately emulate a new user beginning a browsing session. This option resets the HTML context, so that a contextless function is always recorded in the beginning of the action. It also clears the cache and resets the user-names and passwords.

**Save snapshot resources locally:** This option instructs VuGen to save a local copy of the snapshot resources during record and replay. This feature lets VuGen create snapshots more accurately and display them quicker.

**Generate web_reg_find functions for page titles:** (Web, Oracle NCA only) This option enables the generation of **web_reg_find** functions for all HTML page titles. VuGen adds the string from the page's title tag and uses it as an argument for **web_reg_find**.

- 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.

**Add comment to script for HTTP errors while recording** This option adds a comment to the script for each HTTP request error. An error request is defined as one that generated a server response value of 400 or greater during recording.

**Support charset:**

- **UTF-8:** This option enables support for UTF-8 encoding. This instructs VuGen to convert non-ASCII UTF-8 characters to the encoding of your locale's machine in order to display them properly in VuGen. You should enable this option only on non-English UTF-8 encoded pages. The recorded site's language must match the operating system language. You cannot record non-English Web pages with different encodings (e.g. UTF-8 together with ISO-8859-1 or shift_jis) within the same script.
- **EUC-JP:** For users of Japanese Windows, select this option to enable support for Web sites that use EUC-JP character encoding. This instructs VuGen to convert EUC-JP strings to the encoding of your locale's machine in order to display them properly in VuGen. VuGen converts all EUC-JP (Japanese UNIX) strings to the SJIS (Japanese Windows) encoding of your locale's machine, and adds a **web_sjis_to_euc_param** function to the script. (Kanji only)

## Selecting a Recording Engine

By default, VuGen uses the multi-protocol recording engine for all recordings, even if you are only recording a single protocol.

To use the single-protocol recording engine for backward compatibility, select the **Record script using single-protocol recording engine** option in the Advanced Recording Options. If you enable this option, VuGen will use the single-protocol engine the next time you record a session.

# Setting a Recording Scheme

You can further customize the recording by specifying a recording scheme in the following areas:

- Recording Custom Headers
- Filtering Content Type
- Specifying Non-Resource Content Types

### Recording Custom Headers

Web Vusers automatically send several standard HTTP headers with every HTTP request submitted to the server. Click **Headers** to instruct VuGen to record additional HTTP headers. You can work in three modes: **Do not Record Headers**, **Record Headers in list,** or **Record Headers not in list.** When you work in the first mode, VuGen does not record any headers. In the second mode, VuGen only records the checked custom headers. If you specify **Record headers not in list,** VuGen records all custom headers except for those that are checked and other risky headers.

The following standard headers are known as **risky** headers: Authorization, Connection, Content-Length, Cookie, Host, If-Modified-Since, Proxy-Authenticate, Proxy-Authorization, Proxy-Connection, Referer, and WWW-Authenticate. They are not recorded unless selected in the Header list. The default option is **Do not Record Headers**.

In the **Record Headers in List** mode, VuGen inserts a **web_add_auto_header** function into your script for each of the checked headers that it detects. This mode is ideal for recording risky headers that are not recorded unless explicitly stated.

In the **Record Headers not in List** mode, VuGen inserts a **web_add_auto_header** function into your script for each of the unchecked headers that it detects during recording.

To determine which custom headers to record, you can perform a recording session indicating to VuGen to record all headers (see procedure below). Afterwards, you can decide which headers to record and which to exclude.

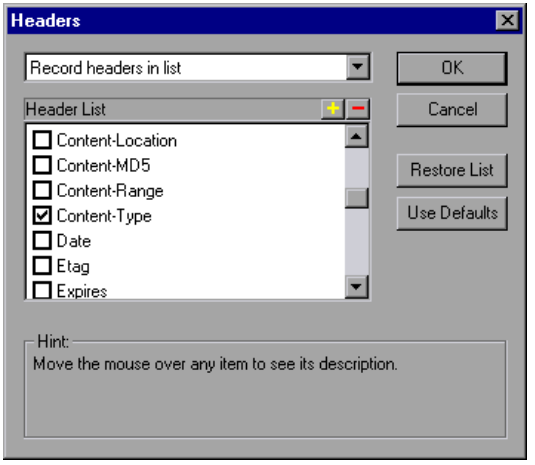
In this example, the **Content-type** header was specified in the **Record Headers in List** mode. VuGen detected the header and added the following statement to the script:

```
web_add_auto_header("Content-Type","application/x-www-form-urlencoded");
```

indicating to the server that the Content-type of the application is x-www-form-urlencode.

**To control the recording of custom headers:**

1. In the Recording Options tree, select the **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

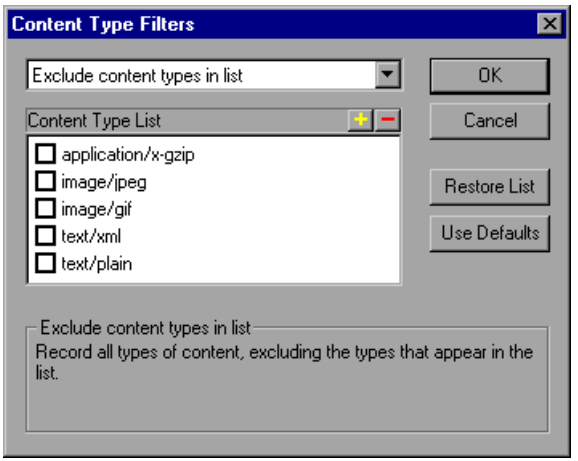
VuGen allows you to filter the content type for your recorded script. You specify the type of the content you wish to record or exclude from your script. You can work in three modes: **Do not Filter Content Types**, **Exclude content types in list,** or **Exclude content types not in list.** When you 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

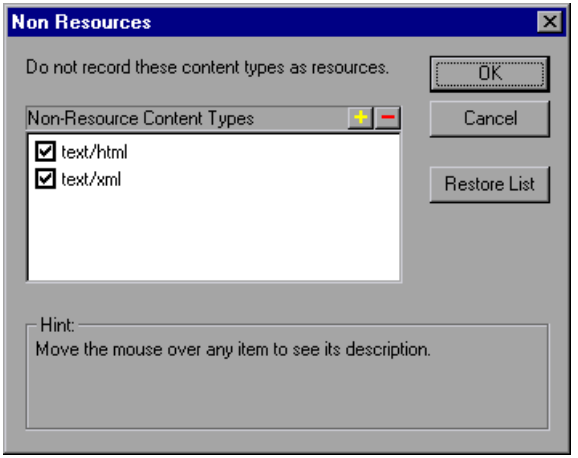
When you record a script, VuGen indicates whether or not it will retrieve the resource during replay using the **Resource** attribute in the **web_url** function. If the **Resource** attribute is set to 0, the resource is retrieved during script execution. If the **Resource** attribute is set to 1, the Vuser skips the resource type.

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

You can exclude specific content types from being handled as resources. For example, you can indicate to VuGen that **gif** type resources should not be handled as a resource and therefore be downloaded unconditionally. When VuGen encounters a **gif** type resource, it sets the **Resource** attribute to 0, indicating to VuGen to download gifs unconditionally during replay.

**To specify which content should not be recorded as resources:**

1. In the Recording Options tree, select the **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.

# 41

# Setting Recording Options for Web Vusers

Before recording a Web session, you can customize the recording options.

This chapter describes:

- About Setting Recording Options
- Specifying which Browser to Use for Recording
- Selecting a Recording Level

***The following information applies to Web and PeopleSoft Enterprise Vuser scripts.***

## About Setting Recording Options

VuGen enables you to generate Web Vuser scripts by recording typical processes that users perform on your Web site.

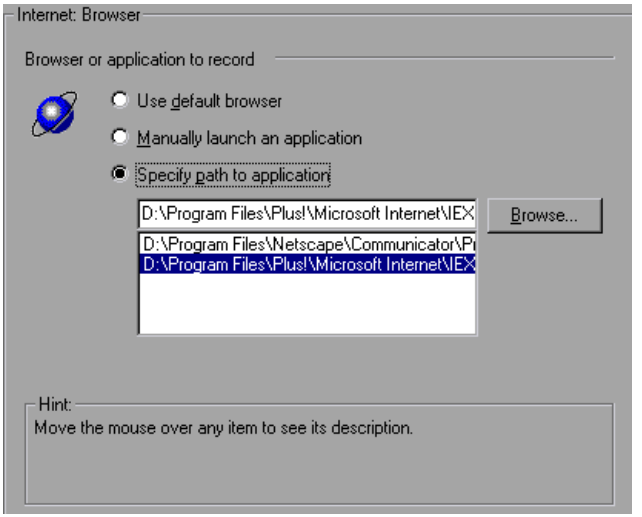
Before recording, you can configure the Recording Options and specify the information to record, the browser or client with which to record, and designate the content for your scripts.

You can set the common Internet protocol recording options, such as proxy settings and other advanced settings. For more information see Chapter 40, "Setting Recording Options for Internet Protocols."

You can also set Correlation recording options for Web Vuser scripts. For more information, see Chapter 46, "Setting 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:

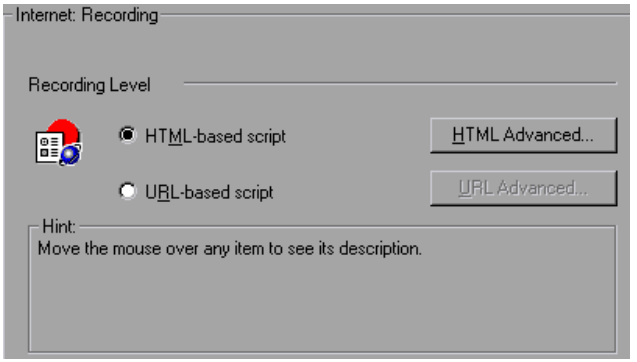
- **Use default browser**, to instruct VuGen to use the default Web browser on the recording computer.
- **Manually launch an application,** to instruct VuGen not to launch a browser when you start recording. You must manually launch a browser or application after you start the recording session.
- **Specify 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 Level

VuGen lets you specify what information to record and which functions to use when generating a Vuser script or Tuning Module session step by selecting a recording level. The recording level you select, depends on your needs and environment. The available levels are **HTML-based script,** and **URL-based script**.

Use the following guidelines to decide which recording level to choose:

- For browser applications without JavaScript, use the HTML-based level.
- For non-browser applications, use the URL-based level.




For PeopleSoft Enterprise and Oracle Web Applications 11i Vusers, there is an additional recording level, **GUI-based**. This option instructs VuGen to generates intuitive context sensitive functions for all user actions. For details, see "Selecting a Recording Level for GUI-Level Vusers" on page 859.

The **HTML-based script** level generates a separate step for each HTML user action. The steps are also intuitive, but they do not reflect true emulation of the JavaScript code.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("MercuryWebTours",
        "URL=http://localhost/MercuryWebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

The **URL-based script** mode option instructs VuGen to record all browser requests and resources from the server that were sent due to the user's actions. It automatically records every HTTP resource as URL steps (**web_url** statements). For normal browser recordings, it is not recommended to use the URL-based mode since is more prone to correlation related issues. If, however, you are recording pages such as applets and non-browser applications, this mode is ideal.

URL-based scripts are not as intuitive as the HTML-based scripts, since all actions are recorded as **web_url** steps instead of **web_link**, **web_image**, and so on.

```
/* URL-based mode - only web_url functions */
…
web_url("spacer.gif",
        "URL=http://graphics.mercury.com/images/spacer.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=",
        "Mode=HTTP",
        LAST);

web_url("calendar_functions.js",
        "URL=http://www.im.mercury.com/travelp/calendar_functions.js",
        "Resource=1",
        "RecContentType=application/x-javascript",
        "Referer=",
        "Mode=HTTP",
        LAST);
…
```

You can switch recording levels and advanced recording options while recording, provided that you are not recording a multi-protocol script. The option of mixing recording levels is available for advanced users for performance tuning.

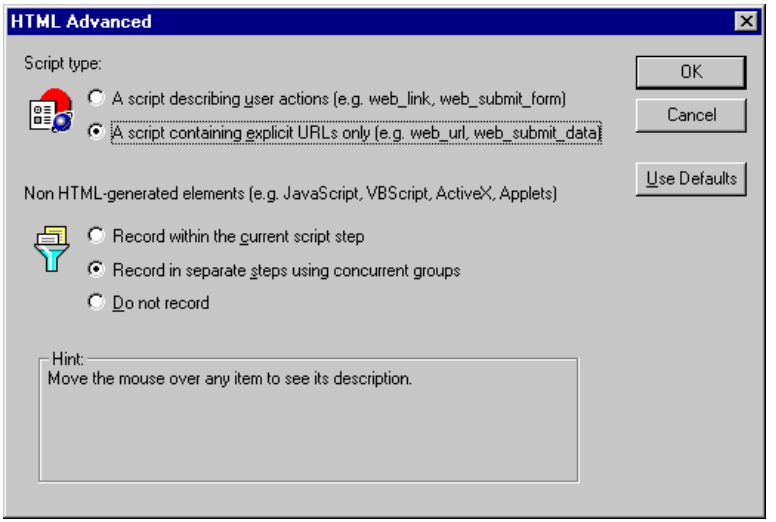
You can also regenerate a script after recording, using a different method than the original recording. For example, if your record a script on an HTML-based level, you can regenerate it on a URL-based level. To regenerate a script, choose **Tools** > **Regenerate Script** and click **Options** to set the recording options for the regeneration.

### Setting Advanced HTML-Based Options

The **HTML-based** option, which is the default recording level, 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 level in the following areas:

- Specifying Script Types
- Handling Non HTML-Generated Elements



### Specifying Script Types

In HTML-based level, 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("MercuryWebTours",
        "URL=http://localhost/MercuryWebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

The second option, **a script containing explicit URLs only**, records all links, images and URLs as **web_url** statements, or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions. The resulting script is less intuitive. This mode is useful for instances where many links within your site have the same link text. If you record the site using the first option, it records an ordinal (instance) for the link, but if you record using the second option, each link is listed by its URL. This facilitates parameterization and correlation for that step.

The following segment illustrates a session recorded with **a script containing explicit URLs only** selected:

```
/* A HTML-based script containing explicit URLs only*//
…
web_url("Click Here For Additional Restrictions",
        "URL=http://www.mercury.com/restrictions.html",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.mercury.com/home?…
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

web_url("buttonhelp.gif",
        "URL=http://www.mercury.com/home?com/rstr?BV_EngineID...,
        "TargetFrame=main",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.mercury.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
…
```

### Handling Non HTML-Generated Elements

Many Web pages contain non-HTML elements, such as applets, XML, ActiveX elements, or javascript. These non-HTML elements usually contain or retrieve their own resources. For example, a javascript **js** file, called from the recorded web page, may load several images. An applet may load an external text file. Using the following options, you can control how VuGen records non HTML-generated elements.

The following options are available:

- Record within the current script step (default)
- Record in separate steps using concurrent groups
- Do not record

The first option, **Record within the current script step**, does not generate a new function for each of the non HTML-generated resources. It lists all resources as arguments of the **web_url**, **web_link**, **web_submit_data**, and so on. statement that was generated for the page. The resources, arguments of the web functions, are indicated by the **EXTRARES** flag. In the following example, the **web_url** function lists all of the non HTML-generated resources loaded on the page:

```
web_url("index.asp",
        "URL=http://www.daisy.com/index.asp",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t2.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=http://www.daisy.com/ScrollApplet.class", "Referer=", ENDITEM,
        "Url=http://www.daisy.com/board.txt", "Referer=", ENDITEM,
        "Url=http://www.daisy.com/nav_login1.gif", ENDITEM,
        …
        LAST);
```

The second option, **Record in separate steps using concurrent groups**, creates a new function for each one of the non HTML-generated resources—it does not include them as items in the page's functions (such as **web_url**, **web_link**, and so on). All of the **web_url** functions generated for a resource, are placed in a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**).

In the following example, the above session was recorded with this option selected. A **web_url** function was generated for the applet and text file loaded with the applet:

```
web_url("index.asp",
        "URL=http://www.daisy.com/index.asp",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t2.inf",
        "Mode=HTML",
        LAST);

web_concurrent_start(NULL);
    web_url("ScrollApplet.class",
        "URL=http://www.daisy.com/ScrollApplet.class",
        "Resource=1",
        "RecContentType=application/octet-stream",
        "Referer=",
        LAST);

    web_url("board.txt",
        "URL=http://www.daisy.com/board.txt",
        "Resource=1",
        "RecContentType=text/plain",
        "Referer=",
        LAST);
web_concurrent_end(NULL);
```

The third option, **Do not record**, instructs VuGen not to record any of the resources generated by non-HTML elements.

Note that when you work in HTML-Based mode, VuGen inserts the **TargetFrame** attribute in the **web_url** statement. VuGen uses this information to display the Web page correctly in the run-time browser and Test Result report.

```
web_url("buttonhelp.gif",
        "URL=http://www.mercury.com/home?com/rstr?BV_EngineID...,
        "TargetFrame=main",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.mercury.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
```

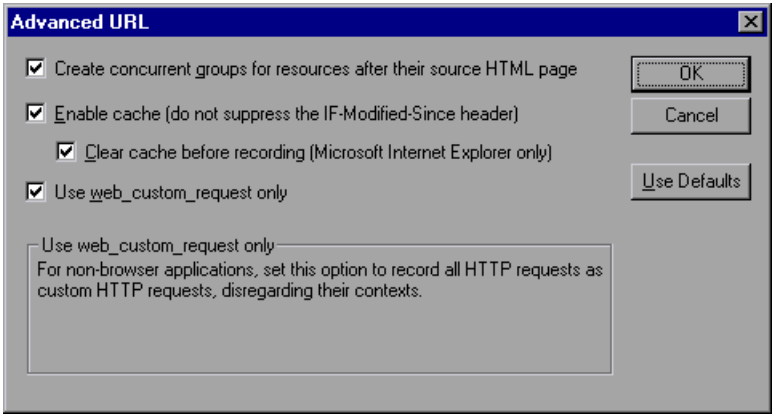
When you record the URL-based mode, VuGen records the content of all frames on the page and therefore omits the TargetFrame attribute.

## Setting Advanced URL-Based Options

The **URL-based** mode option instructs VuGen to record all requests and resources from the server. It automatically records every HTTP resource as URL steps (**web_url** statements), or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions, nor does it record frames.

VuGen lets you set advanced options for the URL recording mode in the following area:

- Resource Handling
- Browser Cache




### Resource Handling

In URL-based recording, VuGen captures all resources downloaded as a result of a browser request. By default, this option is enabled and VuGen records the resources in a concurrent group (enclosed by **web_concurrent_start** and **web_concurrent_end** statements) after the URL. Resources include files such as images, and **js** files. If you disable this option, the resources are listed as separate **web_url** steps, but not marked as a concurrent group.

The following segment illustrates a session recorded with the **Create concurrent groups for resources after their source HTML page** option enabled.

```
web_concurrent_start (NULL);
…
web_url("Click Here For Additional Restrictions",
        "URL=http://www.mercury.com/restrictions.html",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.mercury.com/home?…
        "Snapshot=t4.inf",
        "Mode=HTTP",
        LAST);

web_url("buttonhelp.gif",
        "URL=http://www.mercury.com/home?com/rstr?BV_EngineID...,
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.mercury.com/home?…
        "Snapshot=t5.inf",
        "Mode=HTTP",
        LAST);
…
web_concurrent_end (NULL);
```

Note that the script includes **gif**, and **js** files. This mode also includes other graphic files and imported file such as **imp**, **txt**, 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 the Browser Cache options only apply to single-protocol Web (HTTP/HTML) Vusers—not multi-protocol. Also note that you can manually control this header using the Advanced Header options. (See "Recording Custom Headers" on page 566.)

### 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.mercury.com",
        "URL=http://www.mercury.com/",
        "Method=GET",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTTP",
        LAST);
```

### Enabling EUC-Encoded Web Pages

(This option is only for Japanese Windows.) When working with non-Windows standard character sets, you may need to perform a code conversion. A character set is a mapping from a set of characters to a set of integers. This mapping forms a unique character-integer combination, for a given alphabet. Extended UNIX Code (EUC) and Shift Japan Industry Standard (SJIS) are non-Windows standard character sets used to display Japanese writings on Web sites.

Windows uses SJIS encoding while UNIX uses EUC encoding. When a Web server is on a UNIX machine and the client is Windows, the characters in a Web site are not displayed on the client side properly due to the difference in the encoding methods. This affects the display of EUC-encoded Japanese characters in a Vuser script.

During recording, VuGen detects the encoding of a Web page through its HTTP header. If the information on the character set is not present in the HTTP header, it checks the HTML meta tag. If the page does not send the character set information to the HTTP header or meta tag, VuGen does not detect the EUC encoding.

If you know in advance that a Web page is encoded in EUC, you can instruct VuGen to use the correct encoding during record. To record a page in EUC-encoding, enable the **EUC** option in the Recording Options **Recording** tab**.** **(**only visible for Japanese Windows)

Enabling the **EUC** option, forces VuGen to record a Web page in EUC encoding, even when it is not EUC- encoded. You should, therefore, only enable this option when VuGen cannot detect the encoding from the HTTP header or the HTML meta tag, and when you know in advance that the page is EUC-encoded.

During recording, VuGen receives an EUC-encoded string from the Web server and converts it to SJIS. The SJIS string is saved in the script's Action function. However, for replay to succeed, the string has to be converted back to EUC before being sent back to the Web server. Therefore, VuGen adds a **web_sjis_to_euc_param** function before the Action function, which converts the SJIS string back to EUC.

In the following example, the user goes to an EUC-encoded Web page and clicks on a link. VuGen records the Action function and adds the **web_sjis_to_euc_param** function to the script before the Action function.

```
web_sjis_to_euc_param("param_link","Search");

web_link("LinkStep","Text={param_link}");
```

## Setting the Recording Level

This section describes the procedure for setting the recording levels and their advanced options. Note that you can switch recording levels and advanced recording options while recording, provided that you are recording a single protocol Web (HTTP/HTML) Vuser.

**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. For more information, see "Setting Advanced HTML-Based Options" on page 578.

**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.

**5** For more information about these options, see "Setting Advanced URL-Based Options" on page 584. For users of Japanese Windows, select the **EUC** option to instruct VuGen to use EUC encoding. If you are recording a website whose pages use only the EUC Encoding (Japanese content), select the **EUC** option. VuGen converts the EUC string to SJIS and adds a **web_sjis_to_euc_param** function. If the server sends this information to the browser (in an HTTP header or an HTML Meta tag), you do not need to enable this option.

# 42

# Configuring Internet Run-Time Settings

After you record an Internet protocol Vuser script, you configure its run-time settings.

This chapter describes:

- About Internet Run-Time Settings
- Setting Proxy Options
- Setting Browser Emulation Properties
- Setting Internet Preferences
- Filtering Web Sites
- Obtaining Debug Information
- Performing HTML Compression

***The following information applies to all Internet Protocol Vuser types such as Web, and Wireless.***

## About Internet Run-Time Settings

After developing a Internet protocol Vuser script, you set the run-time settings.

For information about the general run-time settings that apply to all Vusers, see Chapter 12, "Configuring Run-Time Settings." For information about the network speed run-time settings, see Chapter 13, "Configuring Network Run-Time Settings."

The Internet run-time settings let you configure your Internet environment so that Vusers can accurately emulate real users. You can set Interest run-time settings for Proxy, Browser, 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.

Note that you can also modify the run-time settings from the LoadRunner Controller or from the Mercury Tuning Module. For more information, refer to your product's documentation.

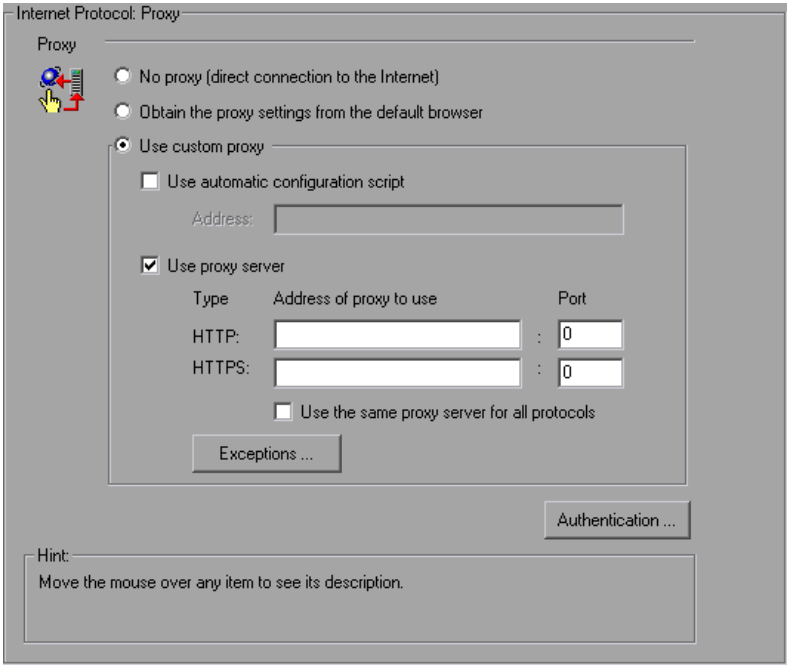
---

**Note:** A run-time setting assigned a value through a Vuser function, overrides the corresponding setting set via the Run-Time Settings dialog box. For more information on using Vuser functions, see Chapter 39, "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, see Chapter 41, "Setting Recording Options for Web Vusers."

The following proxy options are available in the Run-Time settings.

- **No proxy:** All Vusers should use direct connections to the Internet. This means that the connection is made without using a proxy server.
- **Obtain the proxy settings from the default browser:** All Vusers use the proxy settings of the default browser from the machine upon which they are running.

➤ **Use custom proxy:** All Vusers use a custom proxy server. You can supply the actual proxy server details or the path of a proxy automatic configuration script (.**pac** file) that enables automatic configuration. (See "Setting the Automatic Proxy Configuration" on page 595.)

To supply the details of the server, you specify its IP address or name and port. You can specify one proxy server for all HTTP sites, and another proxy server for all HTTPS (secure) sites.

After providing the proxy information, you can specify Authentication information for the proxy server, and indicate Exceptions to the proxy rules.

---

**Note:** To instruct the Vusers to wait for the proxy response during replay, and not to assume that the proxy supports basic authentication, add the following statement:
web_set_sockets_option("PROXY_INITIAL_BASIC_AUTH", "0");

---

## Authentication

If the proxy server requires authentication for each Vuser, use this dialog box to enter the relevant password and user name.

**User Name:** Enter the user name that Vusers will use to access the proxy server.

**Password:** Enter the password required by Vusers to access the proxy server.

---

**Note:** To add authentication dynamically during recording, or to add authentication for multiple proxy servers, use the **web_set_user** function. For more information, see the *Online Function Reference* (**Help > Function Reference**).

---

### Exceptions

You can specify that all Vusers use a specified proxy server. In such a case, if there are any URLs that you want Vusers to access directly, that is, without using the proxy server, enter the list of these URLs in the text box.

**Do not use proxy server for addresses beginning with:** Enter the addresses you want to exclude from the proxy server. Use semicolons to separate entries.

**Do not use proxy server for local (Intranet) addresses:** Select this check box to exclude local addresses, such as those from an intranet, from the proxy server.

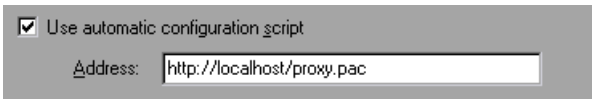
### Setting the Automatic Proxy Configuration

Automatic Proxy Configuration is a feature supported by most browsers. This feature allows you to specify a JavaScript file (usually with a **.pac** extension) containing proxy assignment information. This script tells the browser when to access a proxy server and when to connect directly to the site, depending on the URL. In addition, it can instruct the browser to use a specific proxy server for certain addresses and another server for other addresses.

You can instruct VuGen or your Internet Explorer browser to work with a configuration script. You specify a file for the automatic proxy configuration, so that when the Vuser runs the test, it uses the rules from the proxy file.

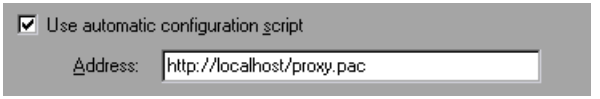
**To specify a configuration script in VuGen:**

1. Choose **Vuser** > **Run-TIme Settings**, and select the **Internet Protocol:Proxy** node.
2. Select **Use custom proxy** and select the **Use automatic configuration script** option. Specify the location of the script.



**To specify a configuration script in Internet Explorer (IE):**

1. Choose **Tools** > **Internet Options**, and select the Connections tab.
2. Click the **LAN Settings** button. The LAN Settings dialog box opens.
3. Select the **Use automatic configuration script** option, and specify the location of the script.




To track the behavior of the Vusers, generate a log during text execution and view the **Execution Log** tab or the **mdrv.log** file. The log shows the proxy servers that were used for each URL. In the following example, VuGen used a direct connection for the URL australia.com, but the proxy server **aqua**, for the URL http://www.google.com.

```
Action1.c(6): t=1141ms: FindProxyForURL returned DIRECT
Action1.c(6): t=1141ms: Resolving australia.com
Action1.c(6): t=1141ms: Connecting to host 199.203.78.255:80
…
Action1.c(6): t=1281ms: Request done "http://australia.com/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
```

## Setting Proxy Run-Time Settings

The following section discusses the steps required for configuring the proxy Run-Time settings.

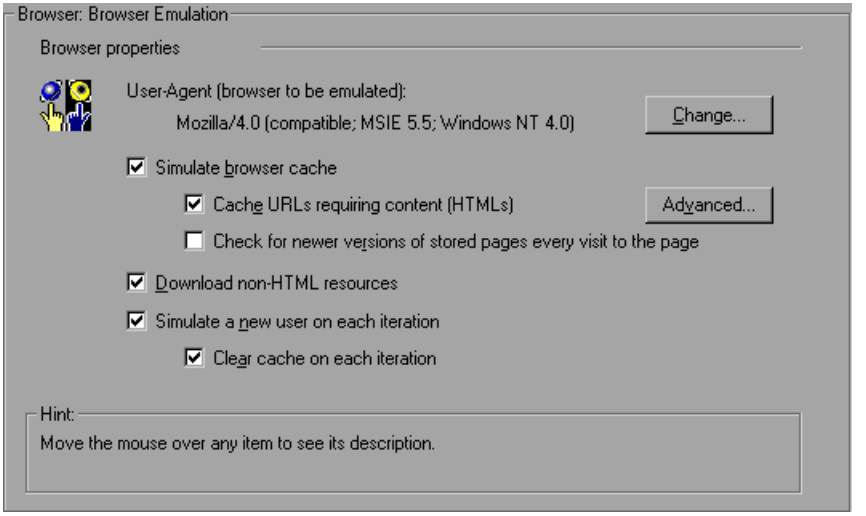
**To set the proxy settings:**



1. Open the Run-Time settings. Click the **Run-Time Settings** button on the VuGen toolbar or choose **Vuser** > **Run-Time Settings**.
2. Click the **Internet Protocol:Proxy** node.
3. Select the desired proxy option: **No proxy, Obtain the proxy settings from the default browser, or Use custom proxy.**
4. If you specified a custom proxy:
   - indicate the IP addresses for the HTTP and HTTPS proxy servers
   - To use a **pac** or javascript file to indicate the proxy, select the **Use automatic configuration script** option and specify the script location. You can specify either a web location beginning with http:// (for example, **http://hostname/proxy.pac**), or a location on the file server, for example, **C:\temp\proxy.pac**.
5. To specify URLs that you want Vusers to access directly, without the proxy server, click **Exceptions** and then supply the list of these URLs. In the Exceptions dialog box, you can also specify direct access to local (intranet) addresses.
6. If the proxy server requires authentication, click **Authentication**, and then supply the relevant password and user name.
7. Select the **Use the same proxy server for all protocols** check box to instruct the Vusers to use the same proxy server for all Internet protocols (HTTP, HTTPS) rather than specifying a specific server for secure sites.

# Setting Browser Emulation Properties

You use the **Browser:Browser Emulation** node in the Run-Time Settings tree to set the browser properties of your tuning or testing environment.




### Browser Properties

You can set the browser properties in the following areas:

- User-Agent (browser to be emulated)
- Simulate browser cache
- Download non-HTML resources
- Simulate a new user each iteration

You can also set advanced options for caching and checking for newer resources.

### User-Agent (browser to be emulated)

Whenever a Vuser sends a request to a Web server, the request includes an HTTP header. The first line of text contains a verb (usually "GET" or "POST"), the resource name (for example "pclt/default.htm"), and the version of the protocol (for example "HTTP/1.0"). Subsequent lines contain "header information" in the form of an attribute name, a colon, and some value. The request ends with a blank line.

All Internet Vuser headers include a **User-Agent** header that identifies the type of browser (or toolkit for Wireless) that is being emulated. For example,

```
User-Agent: Mozilla/3.01Gold (WinNT; I)
```

identifies the Browser as Netscape Navigator Gold version 3.01 running under Windows NT on an Intel machine.

Click **Change...** from the Browser emulation node, to specify the browser information to include in the header. You can specify that a Web Vuser emulate any of the standard browsers. Alternatively, for non-browser HTTP applications, you can specify the HTTP client to match a specific user's application. In this case, you must supply a **Custom User Agent** string that is included in all subsequent HTTP headers. By default, the user-agent emulates the Microsoft Internet Explorer 5.5 browser agent.

#### Simulate browser cache

This option instructs the Vuser to simulate a browser with a cache. A cache is used to keep local copies of frequently accessed documents and thereby reduces the time connected to the network. By default, cache simulation is enabled. When the cache is disabled, Vusers still download each page image only once. When running multiple Vusers as in LoadRunner and Performance Center, every Vuser uses its own cache and retrieves images from the cache. If you disable this option, all Vusers emulate a browser with no cache available.

You can modify your Run-Time settings to match your browser settings for Internet Explorer, as follows:

| Browser Setting | Run-Time Setting |
|---|---|
| Every visit to the page | Select **Simulate Browser Cache** and enable **Check for newer versions of stored pages every visit to the page**. |
| Every time you start Internet Explorer | Select **Simulate Browser Cache** only |
| Automatically | Select **Simulate Browser Cache** only |
| Never | Select **Simulate Browser Cache** and disable **Check for newer versions of stored pages every visit to the page**. |

You can also set the following two browser cache options:

**Cache URLs requiring content (HTML):** This option instructs VuGen to cache only the URLs that require the HTML content. The content may be necessary for parsing, verification, or correlation. When you select this option, HTML content is automatically cached. 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 enabled by default. For more information see "Cache URLs Requiring Content - Advanced" on page 601. If you enable **Simulate browser cache**, but disable this option, VuGen nevertheless stores the graphic files.

**Check for newer versions of stored pages every visit to the page:** This setting instructs the browser to check for later versions of the specified URL, than those stored in the cache. When you enable this option, VuGen adds the "If-modified-since" attribute to the HTTP header. This option brings up the most recent version of the page, but also generates more traffic during the scenario or session execution. By default, browsers do not check for newer resources, and therefore this option is disabled. Configure this option to match the settings in the browser that you want to emulate.

### Download non-HTML resources

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.

**Clear cache on each iteration:** Clears the browser cache for each iteration in order to simulate a user visiting a Web page for the first time. Clear the check box to disable this option and allow Vusers to use the information stored in the browser's cache, simulating a user who recently visited the page.

## Cache URLs Requiring Content - Advanced

The Advanced dialog box lets you specify the URL content types that you want to store in the cache. This dialog box is accessible from the Run-time Settings - **Browser:Browser Emulation** node.

Note that changes to the advanced settings for multiple groups simultaneously, are not supported—edit each group's settings individually.

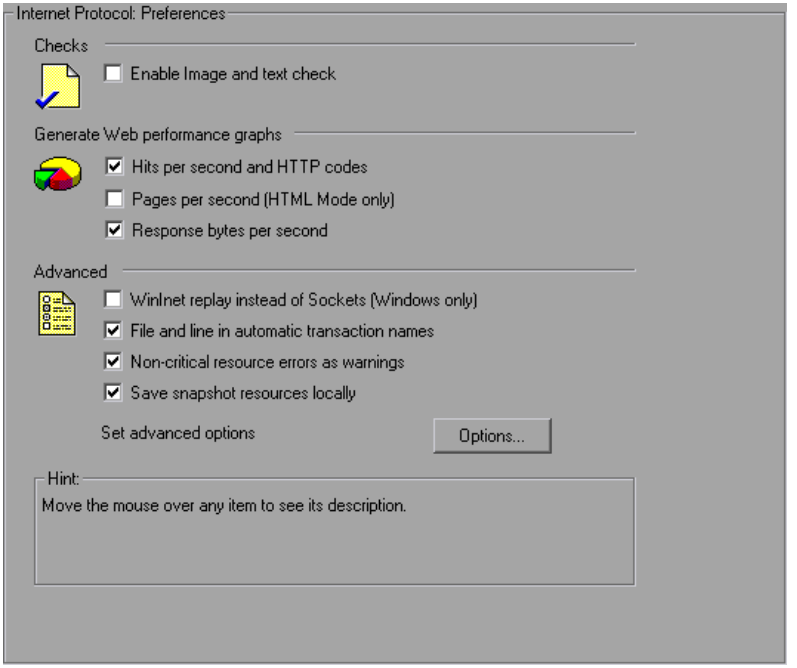
**To add a content type:**

1. Enable the **Specify URLs requiring content in addition to HTML page** option.
2. Click the plus sign to add additional content types, such as text/plain, text/xml, image/jpeg, and image/gif. Enter the content name in the text box.
3. To remove a content type from the list, select it and click the minus sign.

## Setting Internet Preferences

You use the **Internet Protocol:Preferences** node in the Run-Time Settings tree, to set the settings related to the following areas:

- Image and Text Checks
- Generating Web Performance Graphs
- Advanced Web Run-Time Options



### Image and Text Checks

The **Enable image and text checks** option allows the Vuser to perform verification checks during replay by executing the verification functions: **web_find** or **web_image_check**. This option only applies to statements recorded in HTML-based mode. Vusers running with verification checks use more memory than Vusers who do not perform checks (disabled by default).

### Generating Web Performance Graphs

Instructs a Vuser to collect data used to create Web Performance graphs. You view the **Hits per Second**, **Pages per Second,** and **Response Bytes per Second (Throughput)** graphs during test execution using the online monitors and after test execution using the Analysis. You view the Component Breakdown graph after test execution using the Analysis. Select the types of graph data for the Vuser to collect.

---

**Note:** If you do not use the Web performance graphs, disable these options to save memory.

---

### Advanced Web Run-Time Options

**WinInet Replay** (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 load testing. It is also accurate when working with threads. The limitation of the sockets-based engine is that it does not support SOCKS proxy. If you are recording in that type of environment, use the WinInet replay engine.

**File and line in automatic transaction names:** Creates unique transaction names for automatic transactions by adding file name and line number to the transaction name (enabled by default).

---

**Note:** This option places additional information in the log file, and therefore requires more memory.

---

**Non-critical item errors as warnings:** This option returns a warning status for a function which failed on an item that is not critical for load testing, such as an image or Java applet that failed to download. This option is enabled by default. If you want a certain warning to be considered an error and fail your test, you can disable this option. You can set a content-type to be critical by adding it to the list of Non-Resources. For more information, see "Specifying Non-Resource Content Types" on page 570.

**Save snapshot resources locally:** Instructs VuGen to save the snapshot resources to files on the local machine. This feature lets the Run-Time viewer create snapshots more accurately and display them quicker.

## Additional Options for Internet Preferences

Click the **Options** button in the Advanced section of the Preferences node to set advanced options in the following areas: DNS caching, HTTP version, Keep-Alive HTTP connections, Accept server-side compression, Accept-Language headers, HTTP-request connect timeout, HTTP-request receive timeout, Network buffer size, and Step download timeout. The options that only apply to GUI-mode recordings, for PeopleSoft Enterprise and Oracle Applications Vusers, begin with the prefix **GUI-Mode**.

**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" below.
- HTML compression—see "Performing HTML Compression" on page 611.
- 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 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**.

**Accept Server-Side Compression:** Indicate to the server that the replay can accept compressed data. The available options are: **None** (no compression), **gzip** (accept gzip compression), **gzip, deflate** (accept gzip or deflate compression), and **deflate** (accept deflate compression). Note that by accepting compressed data, you may significantly increase the CPU consumption. The default is to accept **gzip, deflate** compression.

**Accept-Language request header:** Provides a comma-separated list of accepted languages. For example, **en-us**, **fr**, and so forth.

**HTTP errors as warnings:** Issues a warning instead of an error upon failing to download resources due to an HTTP error.

**HTTP-request Connect Timeout (seconds):** The time, in seconds, that a Vuser will wait for the connection of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user (default value is 120 seconds). Note that this timeout also applies to the time the Vuser will wait for a WAP connection, initiated by the **wap_connect** function.

**HTTP-request Receive Timeout (seconds):** The time, in seconds, that a Vuser will wait to receive the response of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user (default value is 120 seconds).

The timeout settings are primarily for advanced users who have determined that acceptable timeout values should be different for their environment. The default settings should be sufficient in most cases. If the server does not respond in a reasonable amount of time, check for other connection-related issues, rather than setting a very long timeout which could cause the scripts to wait unnecessarily.

**Step download timeout (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 or Controller, every Vuser uses its own network buffer. This setting is primarily for advanced users who have determined that the network buffer size may affect their script's performance. The default is 12K bytes.

**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 (default is 0).

**Max number of error matches issued as ERRORS:** Limits the number of error matches issued as ERRORS for content checks using a LB or RB (left boundary or right boundary). This applies to matches where a failure occurs when the string is found (Fail=Found). All subsequent matches are listed as informational messages. The default is 10 matches.

**Request Zlib Headers:** Sends request data to the server with the **zlib** compression library headers. By default, requests sent to the server include the **zlib** headers. This option lets you emulate non-browser applications that do not include **zlib** headers in their requests. To exclude these headers, set this option to **No** (default is Yes).

**Enable integrated Authentication:** Enable Kerberos-based authentication. When the server proposes authentication schemes, use **Negotiate** in preference to other schemes. The default is **No**.

**KDC Address:** The address of a Kerberos KDC (Key Distribution Server) which will be used to obtain the TGS (Ticket Granting Service).

**AS Address:** The address of a Kerberos AS (Administration Server) which will be used to obtain information about the principles.

**Maximum number of META Refresh to the same page:** The maximum number of times that a META refresh can be performed per page. The default is 2.

**GUI-Mode default block size for DOM memory allocations:** Sets the default block size for DOM memory allocations. If the value is too small, it may result in extra calls to malloc, slowing the execution times. Too large a block size, may result in an unnecessarily big footprint (default is 16384 bytes).

**GUI-Mode single setTimeout/setInterval threshold (seconds):** Specifies an upper timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, these methods will not invoke the functions that are passed to them. This emulates a user waiting a specified time before clicking on the next element (default is 5 seconds).

**GUI-Mode accumulative setTimeout/setInterval threshold (seconds):** Specifies a timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, additional calls to window.setTimeout and window.setInterval will be ignored. The timeout is accumulative per step (default is 15 seconds).

**GUI-Mode fail on JavaScript error:** Fails the Vuser when a Javascript evaluation error occurs. The default is No, issuing a warning message only after a Javascript error, but continuing to run the script.

**GUI-Mode support ActiveX controls:** Enables the Vusers to execute ActiveX controls (default is No).

**GUI-Mode history support:** Enables support for the window.history object for the test run. The options are Enabled, Disabled, and Auto. The Auto option (default) instructs Vusers to support the window.history object only if it was used in the first iteration. Note that by disabling this option, you improve performance.

**GUI-Mode JavaScript Runtime memory size (KB):** Specifies the size of the JavaScript runtime memory in kilobytes (default is 256 KB).

**GUI-Mode JavaScript Stack memory size (KB):** Specifies the size of the JavaScript stack memory in kilobytes (default is 32 KB).

**GUI-Mode maximum history size:** The maximum number of steps to keep in the history list (default is 10 steps).

**GUI-Mode Home Page URL:** The URL of the home page that opens with your browser (default is about:blank).

**GUI-Mode DOM-based snapshots:** Instructs VuGen to generate snapshots from the DOM instead of from the server responses (Yes by default).

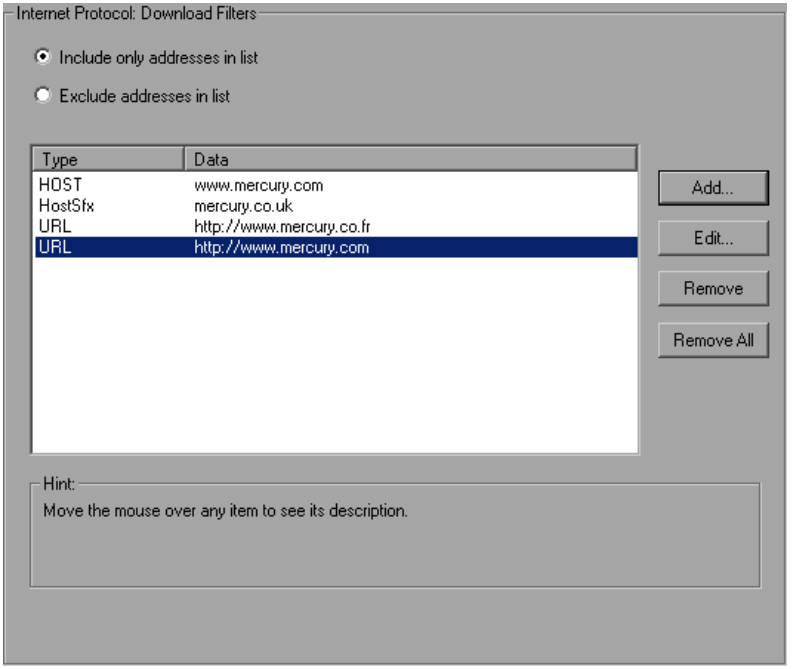
## Filtering Web Sites

You can specify the Web sites from which Vusers should download resources during replay. You can indicate either the sites to exclude or the sites to include. You control the allowed or disallowed sources, by specifying a URL, host name, or host suffix name.

A **URL** is the complete URL address of a Web site, beginning with http:// or https://. **Host** is the name of the host machine with its domain, such as www.mercury.com.

**Host suffix** is the common suffix for several host names, such as mercury.com. This is useful where you have several Web sites on a common domain.

If you specify the sites to exclude, VuGen downloads resources from all Web sites except for those specified in the list. If you specify the sites to include, VuGen filters out resources from all Web sites except for those in the Include list.

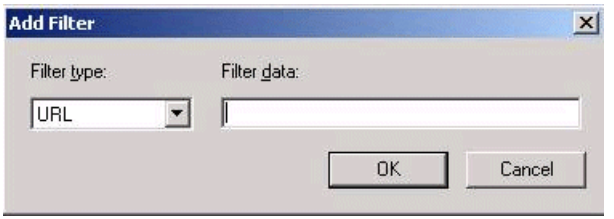



**To create a list of filtered Web sites:**

1 Click the **Internet Protocol:Download Filters** node.

2 Select the desired option: **Include only addresses in list** or **Exclude addresses in list**.

**3** Add entries to the list. To add an entry, click **Add**. The Add filter dialog box opens.




Choose a filter type: **URL**, **Host**, or **Host Suffix**, and enter the filter data, such as a URL. When entering a URL, make sure to enter a complete URL beginning with **http://** or **https://**. Click **OK**.

**4** To edit an entry, select it and click **Edit**.

**5** To delete and entry, select it and click **Remove**. To delete all entries, click **Remove All**.

## Obtaining Debug Information

When you run a Vuser script, the execution information is displayed in the Output window or log file. You control the amount of information sent to the Output window and log files, using the **Log** node of the General run-time settings. For more information, see "Configuring the Log Run-Time Settings" on page 184.

Debug information includes:

- log information
- transaction failures
- the connection status with the gateway—connecting, disconnecting, and redirecting. (WAP only)

To obtain more information for debugging, edit the **default.cfg** file. Locate the **WEB** section and set the **LogFileWrite** flag to **1**. The resulting trace file documents all events in the execution of the script.

When performing load testing, make sure to clear the **LogFileWrite** flag to prevent the Vusers from wasting resources by creating a large trace file.

## Performing HTML Compression

Browsers that support HTTP 1.1 can decompress HTML files. The server compresses the files for transport, substantially reducing the bandwidth required for the data transfer. You can enable compression automatically or manually.

To automatically enable compression in VuGen, use the **Internet Protocol** > **Preferences** node of the Run-Time settings. Click **Options** to open the Advanced Options and enable the **Accept Server-Side compression** option. Note that this option is enabled by default. For more information, see "Additional Options for Internet Preferences" on page 604.

To manually add compression, enter the following function at the beginning of the script:

```
web_add_auto_header("Accept-Encoding", "gzip");
```

To verify that the server sent compressed data, search for the string **Content -Encoding: gzip** in the section of the server's responses of the Execution log. The log also shows the data size before and after decompression.

Compression has a greater effect on large data transfers—the larger the data, the greater effect the compression will have. When working with larger data, you can also increase the network buffer size (see the Network Buffer Size option) to get the data in single chunks.

# 43

# 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:

- About Checking Web Page Content
- Setting the ContentCheck Run-Time Settings

***The following information only applies to Web Vuser types.***

## About Checking Web Page Content

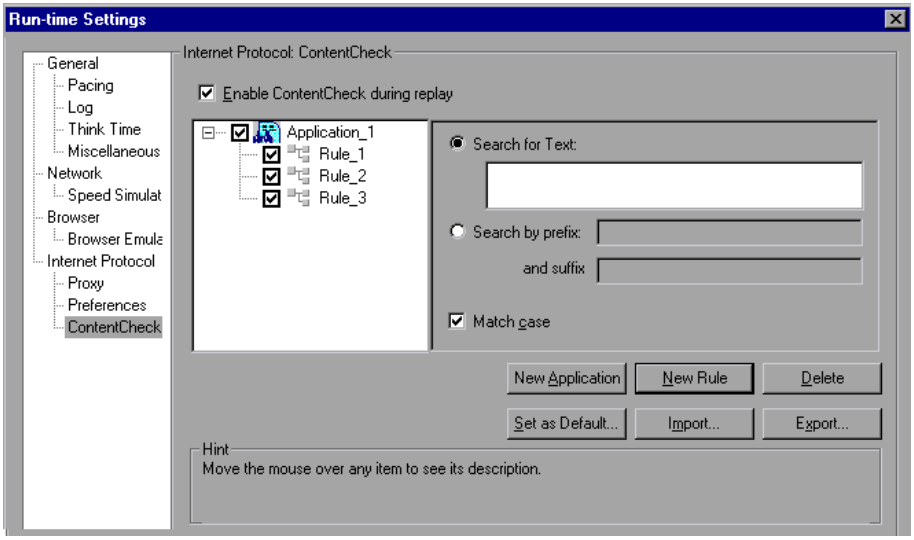
VuGen's Content Check mechanism allows you to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard 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 specify the content for which you want to search. You can define content for several applications with multiple rules. The following sections discuss:

- Understanding Content Rules
- Defining ContentCheck Rules




## Understanding Content Rules

You use the ContentCheck run-time options to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text **ASP Error**. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay. Note that VuGen searches the body of the pages—not the headers.

Note that global changes to ContentCheck settings for multiple groups is not supported—edit each group's settings individually.

**Enable ContentCheck during replay:** Enable content checking during replay. (enabled by default) Note that even after you define applications, you can disable it for a specific test run, by disabling this option.

#### Rule Information

This right pane contains the matching criteria for the text you want to find. You can specify either the actual text or a prefix and suffix of the text.

**Search for Text:** The text of the string for which you want to search.

**Search by Prefix and Suffix:** The prefix and suffix of the string for which you want to search.

**Match case:** Perform a case sensitive search.

**Search JavaScript alert box text:** Only search for text within JavaScript alert boxes. (PeopleSoft Enterprise and Oracle Web Applications 11i Vusers only)

### Adding and Removing Applications and Rules

**New Application:** Automatically adds a new application to the list of applications in the left pane. The default name is Application_**index**, beginning with **Application_1**. After you create a new group, click **New Rule** to add the rule to this group. To modify the name of an application, select the name and click on it.

**New Rule:** Displays the rule criteria in the right pane, allowing you to enter a new rule for the currently selected application. The rules are stored with the script in standard xml files. You can export your rule files and share them with other users or import them to other machines.

**Delete:** Deletes the selected rule or application.

### Importing and Exporting Rules

**Import/Export:** Imports or exports a rule file. The rule file with an **xml** extension, stores the applications and rules. You can export the file to use on other machines. You can also import other rule files. If you import a rule and the selected rule conflicts with an existing rule, VuGen issues a warning indicating that it is a Conflicting Rule. You can then choose to merge the rules you created on a former script with the one you are importing or overwrite the current rules. When you click Export, VuGen opens the Choose Application to Export dialog box.

### Setting Rules as Default

**Set as Default:** There are three types of rules for Content Checks: **Installation**, **Default**, and **per script**. Installation rules are provided automatically during installation of the product. Default rules, apply to all scripts executed on your machine. The per script rules are the ones defined for the current script. When you modify or add rules, these changes only apply to the current script. To instruct VuGen to add a rule to the list of Default rules so that it will apply to all scripts on that machine, click **Set as Default**.

When working on multiple scripts, or when performing a product upgrade, a conflict may arise between the default rules and the script rules. VuGen asks you if you want to merge the rules. When you merge the rules (recommended), the rule is added to the list of rules for the application.

This action only effects applications that are enabled in the Application list (the left pane). If no applications were marked as Enabled in the current script, no application will be marked as Enabled in the Defaults file. Click **Yes** to overwrite the Defaults file. Click **No** to cancel the operation and retain the original Defaults file.

The rules are stored in standard xml files. You can export your rule files and share them with other users or import them to other machines.

**When you click Set as Defaults (and confirm the overwriting), VuGen performs the following actions:**

1. Marks all applications in the Defaults File as **Disabled**.
2. For applications marked as **Enabled** in the current script, it performs a merge or copy, depending on whether the application exists. If the application exists, it merges the rules of the current script with those of the Defaults file. If the application did not exist in the Defaults file, then VuGen just copies the rules to the Defaults file.
3. Marks the applications that were enabled in the script, as **Enabled** in the Defaults file. If no application is marked as **Enabled** in the current script, no application will be marked as **Enabled** in the Defaults file.

#### Use Defaults

Imports rules from the Defaults file. When you click this button, VuGen opens a dialog box with a list of the applications and their default settings. You can choose to import these rules or modify them. If this conflicts with one of the existing rules, VuGen issues a warning indicating that it is a Conflicting Rule. You can also merge the rules defined in the Defaults file with the ones currently defined.

To use the default settings for all of your applications, click **Use Defaults** which imports the definitions from the Defaults file. It opens a dialog box with a list of the applications and their default settings. You can 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 ContentCheck Rules

You use the **Internet Protocol:ContentCheck** node in the Run-Time Setting tree, to define the rules for checking Web page content.

**To define a ContentCheck rule:**

1. Open the Run-Time settings and select the **Internet Protocol:ContentCheck** node.
2. Select the **Enable ContentCheck during replay** option.

3. Click **New Application** to add a new entry to the list of applications whose content to check.
4. Click **New Rule** to add rules for existing applications. Each application server may have one or more rules. Enable or disable the relevant rules by clearing or selecting the check boxes adjacent to the rule in the left pane.
5. To search for the actual text string, select **Search for Text** and specify the text for which you want to search. 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.
6. To search for the text preceding and following your string, select **Search by Prefix** and specify the prefix and suffix.
7. To indicate a case sensitive search, select the **Match case** check box.
8. To set a rule as a default, indicating that it should apply to all scripts on that machine, select the rule or application and click **Set as Default**.
9. To export the rule file click **Export** and specify a save location.
10. To import a rule file, click **Import** and locate the file.
11. To remove an application or rule, select it and click **Delete**.
12. To use the default settings for all of your applications, click **Use Defaults.** A dialog box opens with a list of the applications and their default settings. You can choose to overwrite or merge the rules if there are conflicts.

# 44

# 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:

- About Verification Under Load
- Adding a Text Check
- Understanding Text Check Functions
- Adding an Image Check
- Defining Additional Properties

***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 or an image.

Web checks enable you to determine whether or not your Web site is functioning correctly while it is being accessed by many Vusers—that is, does the server return the correct Web pages? This is particularly important while your site is under the load of many users, when the server is more likely to return incorrect pages.

For example, assume that your Web site displays information on the temperatures in major cities around the world. You use VuGen to create a Vuser script that accesses your Web site.

The Vuser accesses the site and executes a text check on this Web page. For example, if the word **Temperature** appears on the page, the check passes. If **Temperature** does not appear because, for example, the correct page was not returned by the server, the check fails. Note that the text check step appears before the URL step. This is because VuGen registers, or prepares in advance, the search information relevant for the next step. When you run the Vuser script, VuGen conducts the check on the Web page that follows.




Although the server may display the correct page when you record the script and when a single Vuser executes the script, it is possible that the correct page will not be returned when the server is under the load of many Vusers. The server may be overloaded and may therefore return meaningless or incorrect HTML code. Alternatively, in some instances when a server is overloaded, the server may return a **500 Server Error** page. In both of these cases, you can insert a check to determine whether or not the correct page is returned by the server.

---

**Note:** Web checks increase the work of a Vuser, and therefore you may need to run fewer Vusers per load generator. You should use Web checks only where experience has shown that the server sometimes returns an incorrect page.

---

You can define Web checks during or after recording a Vuser script. It is generally more convenient to define checks while recording—when the Web page that you want to check is visible.

VuGen uses several different Web check icons, each one representing a different check type:

| Web Check Icon | Description |
|---|---|
| Text  | A text check, searching for a specific string in the next action function (**web_reg_find**) or in the entire business process (**web_global_verification**) step. |
| Text  | A text check, searching for a specific string in the next action function using the **web_find** step. For more information, see "Understanding Text Check Functions" on page 624. |
| Image  | An image check, searching for a specific image on a Web page. For more information, see "Understanding Text Check Functions" on page 624. |

This chapter describes how to use VuGen to add Web checks in the tree view. For information about adding checks to the script in the text-based script view, see the *Online Function Reference* (**Help > Function Reference**).

# Adding a Text Check

VuGen allows you to add a check that searches for a text string on a Web page. You can add the text check either during or after recording.

When you create a text check, you define the name of the check, the scope of the check, the text you want to check for, and the search conditions.

**To add a text check during recording:**

1 If the VuGen main window or application is minimized, restore it. In the application or Web browser window, select the desired text.



2 Click the **Insert Text check** button on the recording toolbar. VuGen adds a **web_reg_find** function to the script.

**To add a text check after recording:**

1 Go to the snapshot of the step whose text you want to check.

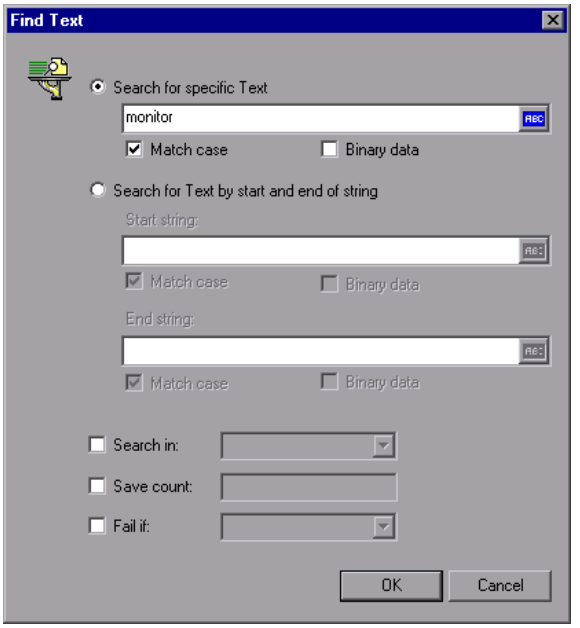
2 In the snapshot, select the text you want to verify.

3 Choose **Add a Text Check (web_reg_find)** from the right-click menu. The Find Text properties dialog box opens.

---

**Note:** For certain protocols, VuGen issues a message indicating that you should add text checks from the Server Response tab—not from the snapshot. Click the **Server Response** tab and select the **HTML Document** tab. Expand the body node and then continue as described below.

---




The following attributes are available for **web_reg_find**:

- **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".

If you do not have a specific string for the **Text**, you can enter values for 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.

To view or modify the properties of the text check after it has been created, click the **Tree View** tab and double-click on the new **Services: Reg Find** step. In the Find Text dialog box, you can view or modify all of the step's attributes.

## Understanding Text Check Functions

When you add a text check, VuGen adds a **web_reg_find** function to your script. This function registers a search for a text string on an HTML page. Registration means that it does not execute the search immediately—it performs the check only after executing the next Action function, such as **web_url**. Note that if you are working with a concurrent functions group, the **web_reg_find** function is only executed at the end of the grouping.

In the following example, **web_reg_find** function searches for the text string "Welcome". If the string is not found, the next action function fails and the script execution stops.

```
web_reg_find("Text=Welcome", "Fail=Found", LAST);
web_url("Step", "URL=...", LAST);
```

In addition to the **web_reg_find** function, you can use other functions to search for text within an HTML page:

Several additional functions can be used for searching for text:

- web_find
- web_global_verification

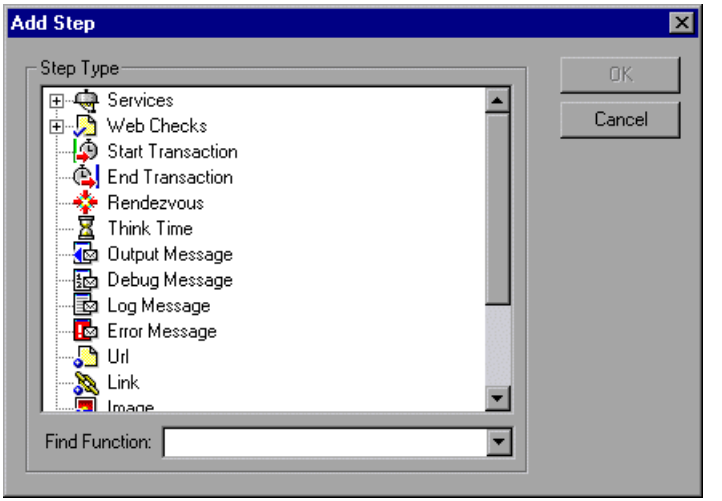
The **web_find** function, primarily used for backward compatibility, differs from the **web_reg_find** function in that **web_find** is limited to an HTML-based script (see **Recording Options** > **Recording** tab). It also has less attributes such as instance, allowing you to determine the number of times the text appeared. When performing a standard text search, **web_reg_find** is the preferred function.

The **web_global_verification** function allows you to search the data of an entire business process. In contrast to **web_reg_find**, which only applies to the next Action function, this function applies to **all** subsequent Action functions such **web_url**. By default, the scope of the search is NORESOURCE, searching only the HTML body, excluding headers and resources.

The **web_global_verification** function is ideal for detecting application level errors that are not included the HTTP status codes. This function is not limited to an HTML-Based script (see **Recording Options** > **Recording** tab).

**To add additional functions to your script:**

1 In the VuGen main window, click at the point where you want to add the text check. Choose **Insert** > **New Step**.



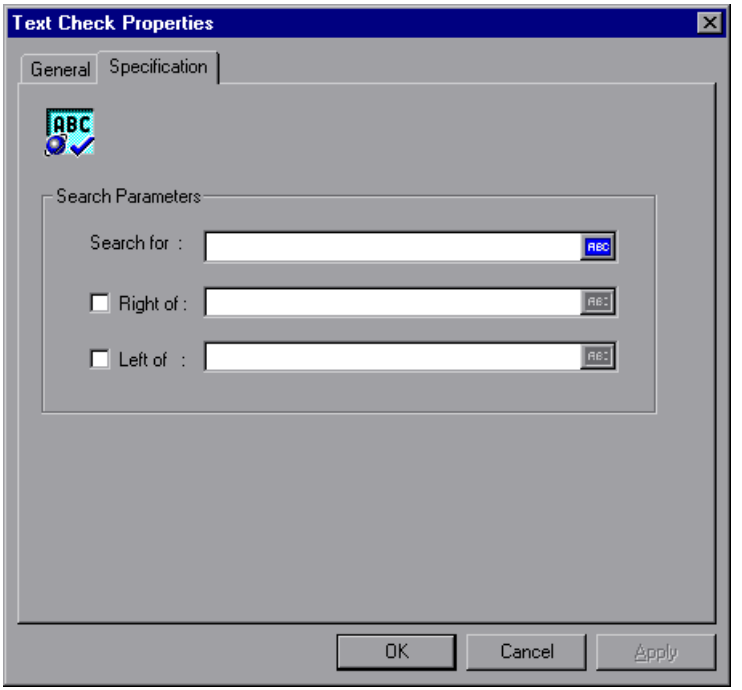

2 For the **web_find** functions, expand the **Web Checks** node and select **Text Check**. For the **web_global_verification** function, expand the **Services** node and choose the function name. The Properties dialog box opens.

3 Set the properties for these functions (see description below).

4 Click **OK**. VuGen inserts a new function into the script.

### Setting web_find Properties

You can set the following properties for the **web_find** function:




**Search for**: the string you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see Chapter 8, "Working with VuGen Parameters."

**Right of** / **Left of:** the position of the search string relative to adjacent text. Type the text in the appropriate field. For example, to verify that the string "support@mercuryinteractive.com" appears to the right of the word "e-mail:," select **Right of** and then type "e-mail:" in the **Right of** box.

**Step Name:** the name of the text check. Click the **General** tab and type a name for the text check in the box. Use a name that you can recognize and identify later on.

---

**Note:** A Vuser conducts Web checks during script execution only if checks are enabled, and if the script runs in HTML mode. To enable checks, select the **Enable image and text check** option in the **Preferences** tab in the Run-Time Settings dialog box. For details, see Chapter 12, "Configuring Run-Time Settings."

---

### Setting web_global_verification Properties

You can set the following properties for the **web_find** function:




**Search for specific text:** the string whose presence you want to verify. An ABC icon indicates that the string in the **Search for** box has not been assigned a parameter. For details on assigning parameters, see Chapter 8, "Working with VuGen Parameters."

**Search for Text by Start and End of String:** the boundaries, also known as **Start** and **End** strings that surround the text. Select the appropriate options to indicate if you want to **Match case** or if you are searching for **binary data**.

**Fail if:** Fails the script if the condition is met. You can also indicate the failure condition: if the text is **Found** or **Not found**. Select the desired behavior in the **Fail if** box.

### Text Flags

When specifying search text using a registered search, **web_reg_find**, you can add flags to control the scope of the search:

**/IC** to ignore the case.

**/BIN** to specify binary data.

**/DIG** to interpret the pound sign (#) as a wildcard for a single digit. The DIG flag does not match a literal pound sign.

**/ALNUM**<case> to interpret the caret sign (^) as a wildcard for a single US-ASCII alphanumeric character. There are three syntaxes: **ALNUMIC** to ignore case, **ALNUMLC** to match only lower case, and **ALNUMUC** to match only upper case. The ALNUM flag does not match a literal caret.

To use flags, you enter the attribute **TEXT**, followed by a forward slash and the flag name. For example, to search for a string ignoring the case, use "Text/IC=search_text'".

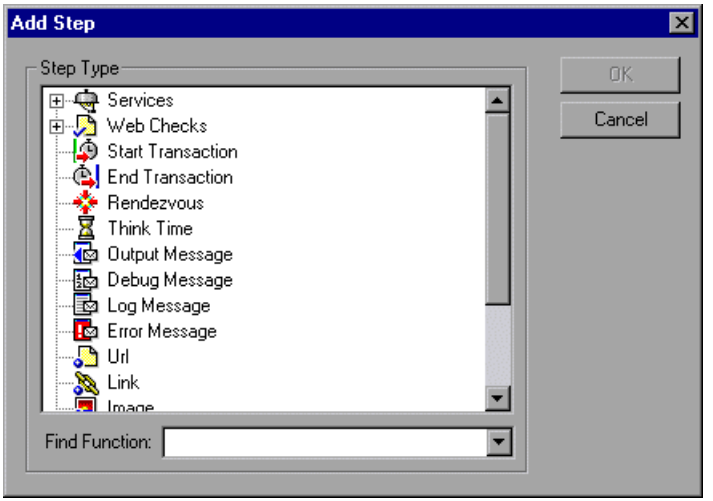
# Adding an Image Check

VuGen allows you to add a user-defined check that searches for an image on a Web page. The image can be identified by the ALT attribute, the SRC attribute, or both.

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.



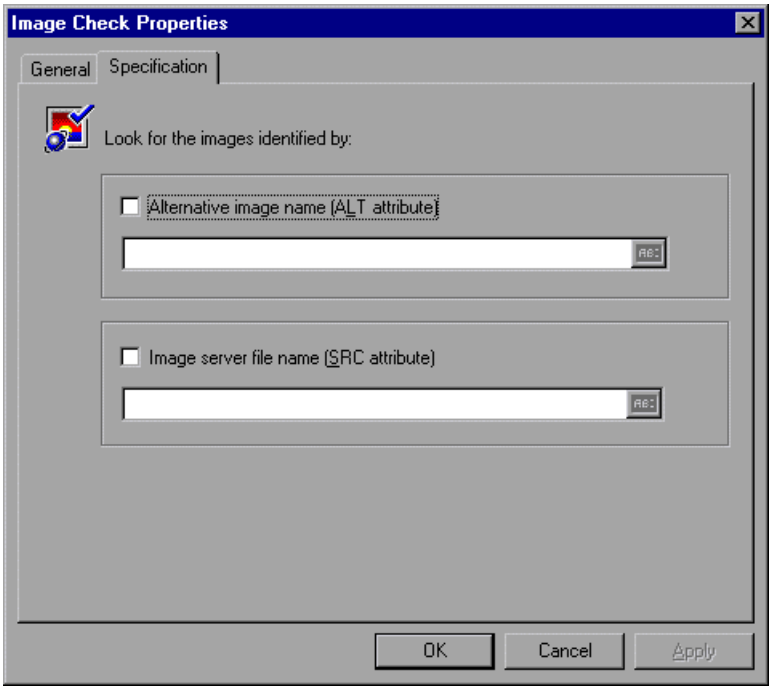

---

**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.



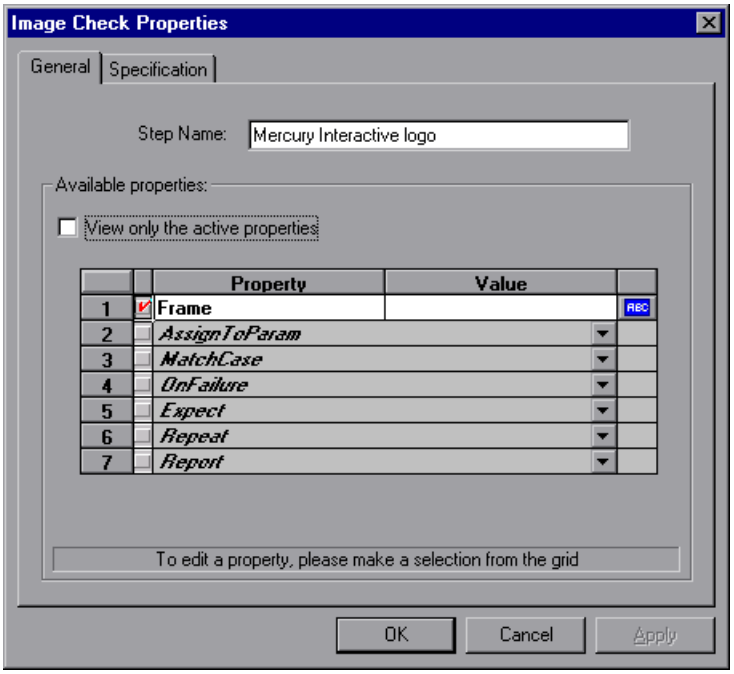

**4** Select a method to identify the image:

- 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 8, "Working with VuGen Parameters."

**Note:** If you select both the ALT attribute and SRC attribute check boxes, the Vuser searches for an image that has both the specified ALT attribute and the specified SRC attribute.

**5** To name the image check, click the **General** tab. In the **Step Name** box, type a name for the image check. Use a name that you can recognize later on.



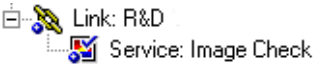

**6** The properties table displays additional properties that define the check.

Clear the **View only the active properties** check box to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column.

For details on assigning property values, see “Defining Additional Properties” on page 633.



**7** Click **OK** to accept the settings. An icon representing the new image check is added to the associated step in the Vuser script.




## Defining Additional Properties

You can specify additional properties for each Web check that you insert into a Vuser script. You set additional options in the properties table on the **General** tab of the check properties dialog boxes. The following is only relevant for **web_find** and **web_image_check** functions—not **web_reg_find**.

**To set additional properties:**

**1** Right-click the Web check whose properties you want to edit, and select **Properties** from the pop-up menu. The appropriate check properties dialog box opens. 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.
- **AssignToParam:** Select **YES** to enable assigning to a parameter. Select **NO** to disable this capability. The default value is **NO**.
- **MatchCase:** Select **YES** to conduct a case-sensitive search. Select **NO** to conduct a non-case-sensitive search. The default value is **NO**.

- **OnFailure:** Select **Abort** to abort the entire Vuser script if the check fails. VuGen aborts the Vuser script regardless of the error-handling method that has been set in the run-time settings. Select **Continue** to have the error-handling method defined in the run-time settings determine whether or not the script is aborted if the check fails.

  The default value is **Continue**. For details on defining the error handling method, see Chapter 12, "Configuring Run-Time Settings."

- **Expect:** Select **NotFound** to indicate that the check is successful if the Vuser does not find the specified check object. Select **Found** to indicate that the check is successful if the Vuser finds the specified check object. The default value is Found.

- **Repeat:** Select **YES** to search for multiple occurrences of the specified check object. Select **NO** to end the check as soon as one occurrence of the specified check object is found. The Vuser script continues with the next step. This option is useful when searching through a Web page that may have multiple occurrences of the check object. The default value is YES.

- **Report:** Select **Always** to always view a detailed description of the check results in the Execution Log. Select **Failure** to view detailed check results only when the check fails. Select **Success** to view detailed check results only when the check succeeds. The default value is **Always**.

An ABC icon indicates that the property value has not been assigned a parameter. Click the icon to assign a parameter. For more information, see Chapter 8, "Working with VuGen Parameters."

# 45

# 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:

- About Modifying Web and Wireless Vuser Scripts
- 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 38, "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, refer to 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.

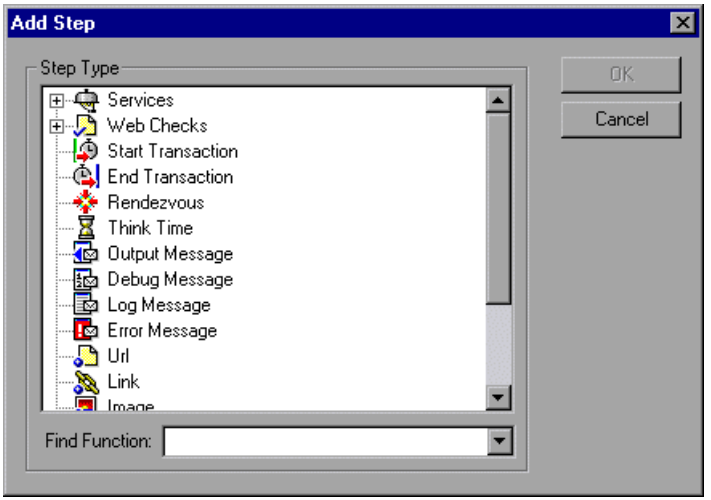
# Adding a Step to a Vuser Script

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 this... | See... |
|---|---|
| Vuser API function | Chapter 7, "Enhancing Vuser Scripts" |
| Service step | "Modifying Service Steps" on page 659 |
| Web Check | "Modifying Web Checks (Web only)" on page 660 |
| Transaction | "Modifying a Transaction" on page 656 |
| Rendezvous point | "Modifying a Rendezvous Point" on page 657 |
| Think time step | "Modifying Think Time" on page 658 |
| URL step | "Modifying a URL Step" on page 639 |
| Link step | "Modifying a Hypertext Link Step (Web only)" on page 641 |
| Image step | "Modifying an Image Step (Web only)" on page 643 |
| Submit form step | "Modifying a Submit Form Step (Web only)" on page 645 |
| Submit data step | "Modifying a Submit Data Step" on page 649 |
| Custom request step | "Modifying a Custom Request Step" on page 653 |
| User-defined step | Chapter 7, "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 Level" on page 575.

### Setting the Replay Mode

In the URL step's Properties dialog box, you can modify the mode settings to instruct Vusers to execute the script in a mode other than the recorded mode. To customize the replay mode, select the **Record mode** check box. The available replay modes are:

**HTML:** Automatically download all resources and images and store the required HTTP information for the steps that follow. This is ideal for script with Web links.

**HTTP:** Do not download any resources for this step during replay. Download only resources that are explicitly represented by functions.

You can also indicate that a certain step should not be counted as a resource. For example, if you have a step that represents a specific image that you want to skip, you can instruct VuGen to exclude that resource type. For more information, see the "Resource Handling" on page 584.

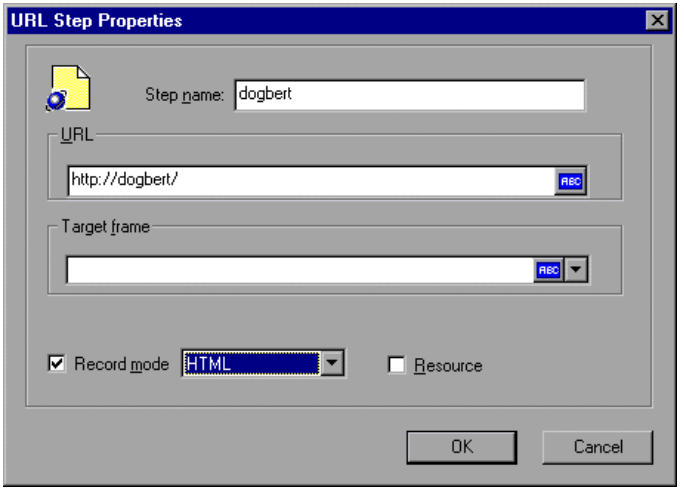
**To modify the properties of a URL step:**



**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 8, "Working with VuGen 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, see Chapter 41, "Setting Recording Options for Web Vusers."

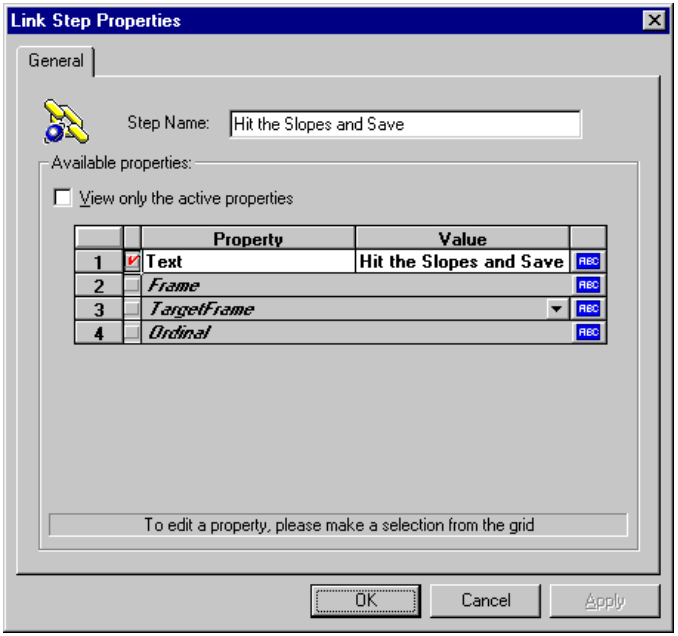
The properties that you can modify are the name of the step, how the hypertext link is identified, and where it is located.

**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. Refer to 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 8, "Working with VuGen Parameters."

**5** Click **OK** to close the Link Step Properties dialog box.

### Modifying an Image Step (Web only)

An image step is added to the Vuser script when you click a hypergraphic link. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. For more information, see Chapter 41, "Setting Recording Options for Web Vusers."

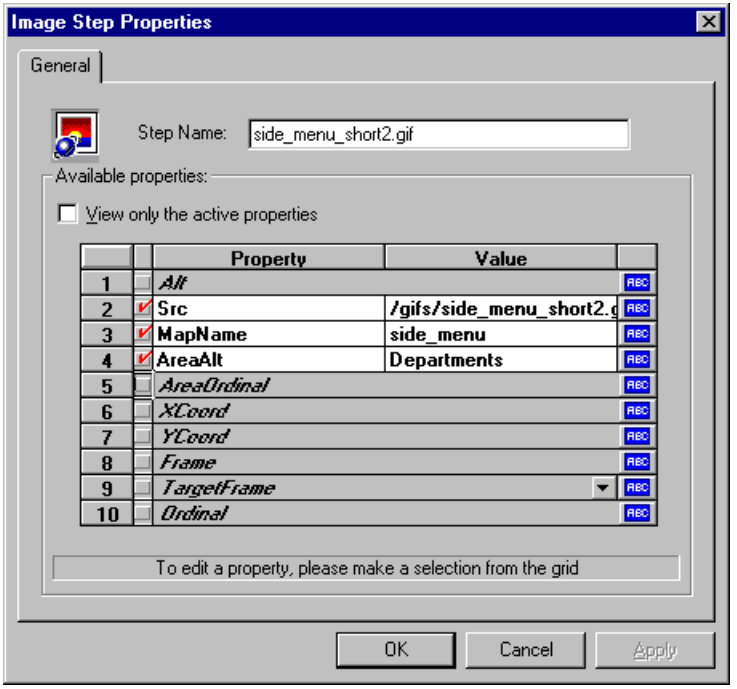
The properties that you can modify are the name of the step, how the hypergraphic link is identified, and where it is located.

**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. Refer to 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 8, "Working with VuGen Parameters."

**5** Click **OK** to close the Image Step Properties dialog box.

### Modifying a Submit Form Step (Web only)

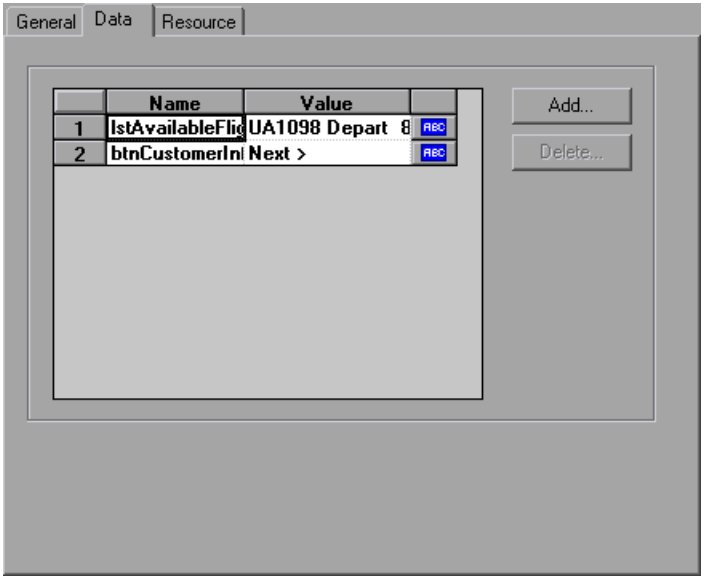
A submit form step is added to the Vuser script when you submit a form. This step is only recorded when you select the option to record in HTML (context-sensitive) mode. For more information, see Chapter 41, "Setting Recording Options for Web Vusers."

The properties that you can modify are the name of the step, the form location, how the form submission is identified, the form data, and the resources for the step.

**To modify the properties of a submit form step:**

1 In the tree view of the Vuser script, select the submit form step you want to edit. Submit form steps are shown using the **Submit Form** icon.

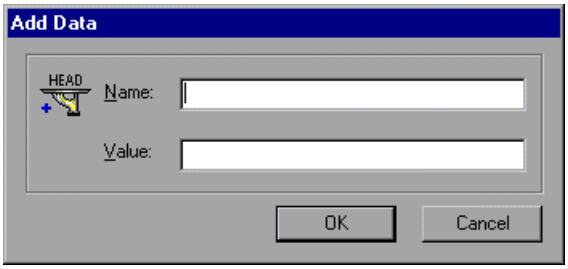
2 Select **Properties** from the right-click menu. The Submit Form Step Properties dialog box opens. Click the **Data** tab.




- The **Name** column lists all the data arguments on the form.
- The **Value** column displays the corresponding value input for a data argument.
- The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 8, “Working with VuGen Parameters,” the ABC icon changes to a table icon.

3 To edit a data argument, double-click on it to activate the cursor within the cell and type the new value in the editable box.

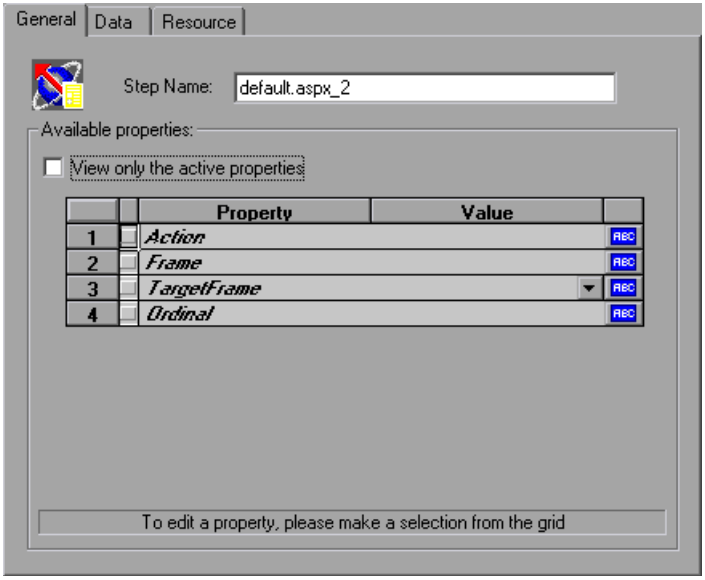
**4** To add a new data argument to the form submission, click **Add**. The Add Data dialog box opens.




**5** Type a **Name** and **Value** for the data argument, and click **OK**.

**6** To delete an argument, select it and click **Delete**.

**7** To change the name of the submit form step, click the **General** tab.




**8** To change the step name, type a new name in the **Step Name** box. The default name during recording is the name of the executable program used to process the form.

**9** The properties table displays the properties that identify the form submission.

Clear the **View only the active properties** option to view active and non-active properties. To enable a property, click the cell to the left of the property name. Assign the property a value in the **Value** column:

- **Action:** the address to be used to carry out the action of the form
- **Frame:** the name of the frame where the form submission is located
- **TargetFrame:** the target frame:

  **_TOP:** replaces the whole page

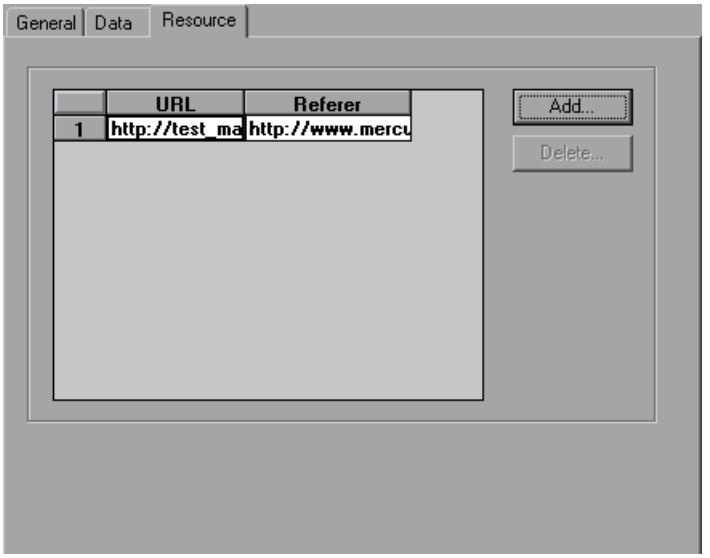
  **_BLANK:** opens a new window

  **_PARENT**: replaces the parent of the last (changed) frame

  **_SELF:** replaces the last (changed) frame
- **Ordinal**: a number that uniquely identifies the form when all other property attributes are identical to one or more other forms on the same Web page. Refer to 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 8, "Working with VuGen Parameters."

**10** To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.




**11** Click **OK** to close the Submit Form Step Properties dialog box.

### Modifying a Submit Data Step

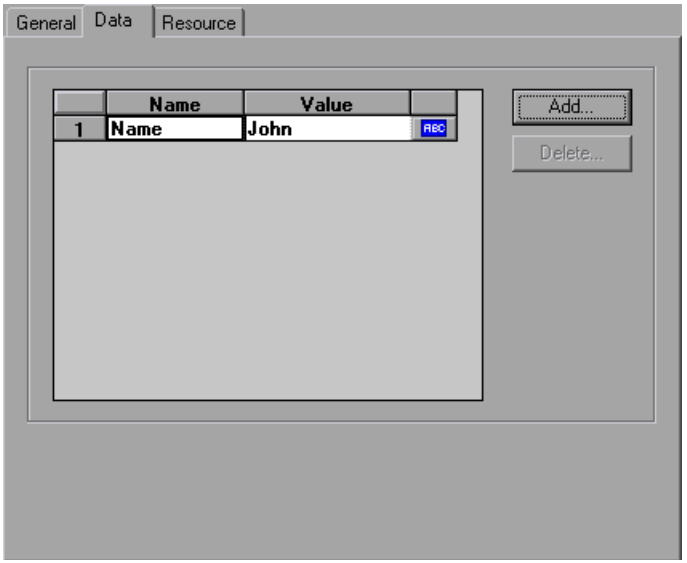
A submit data step represents the submission of a form of data to your Web site for processing. This is different from a Submit Form step because you do not need to have a form context to execute this request.

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 represented by the **Submit Data** icon.

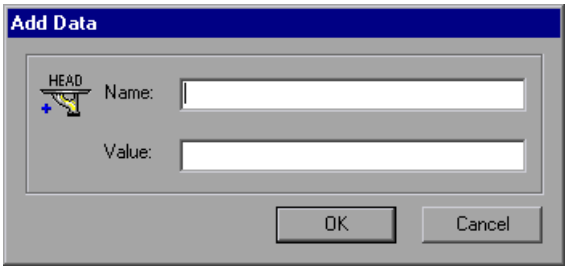
**2** Select **Properties** from the right-click menu. The Submit Data Step Properties dialog box opens. Click the **Data** tab.




- The **Name** column lists all the data arguments on the form. This includes all hidden fields.
- The **Value** column displays the corresponding value input for a data argument.
- The type column contains an icon. Initially, all values are constants or non-parameterized values and have an ABC icon. If you assign a parameter to the data value, as described in Chapter 8, "Working with VuGen Parameters," the **ABC** icon changes to a table icon.

**3** To edit a data argument, double-click on it to activate the cursor within the cell. Then type the new value.

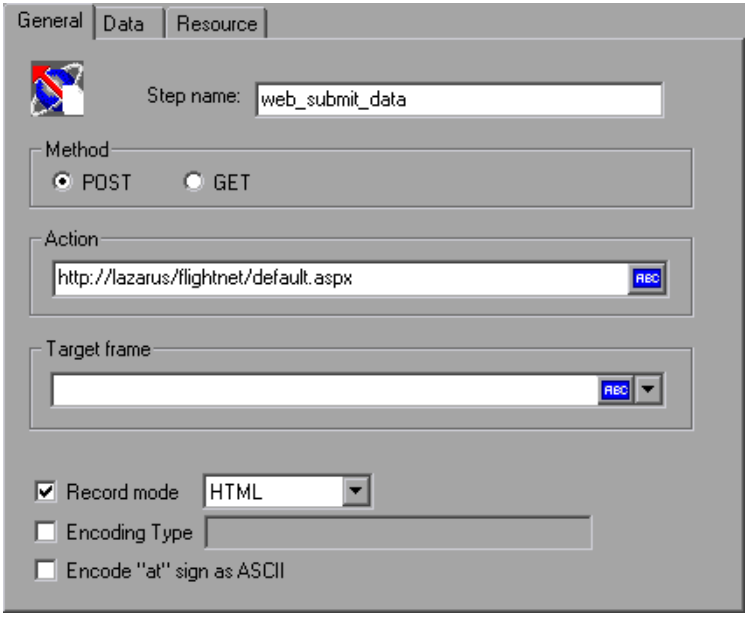
**4** To add new data, click **Add**. The Add Data dialog box opens.




**5** Type a **Name** and **Value** for the data argument, and click **OK**.

**6** To delete an argument, select it and click **Delete**.

**7** To change the name of the submit data step, click the **General** tab.




**8** To change the step name, type a new name in the **Step name** box.

**9** Under **Method**, click **POST** or **GET**. The default method is **POST**.

**10** In the **Action** box, type the address to be used to carry out the action of the data submission. An ABC icon indicates that the action has not been assigned a parameter. For details on assigning parameters, see Chapter 8, "Working with VuGen Parameters."

**11** Select a **Target frame** from the list:

**_TOP:** replaces the whole page

**_BLANK:** opens a new window

**_PARENT**: replaces the parent of the last (changed) frame

**_SELF:** replaces the last (changed) frame

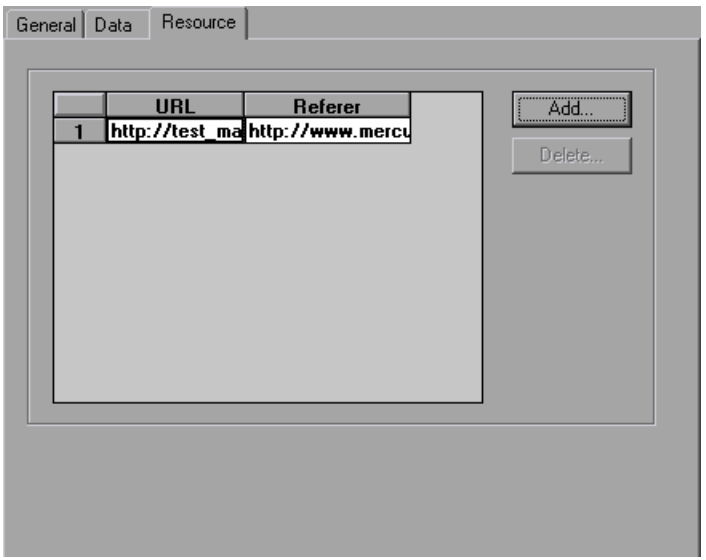
**12** To customize the replay mode, select the **Record mode** option. Choose the desired mode: HTML, or HTTP. For more information, see "Setting the Replay Mode" on page 640.

**13** To specify an encoding type, such as **multipart/www-urlencoded**, select the **Encoding type** check box and specify the encoding method.

**14** To encode the "@" in the URL, select **Encode "at" sign as ASCII**.

**15** Click **OK** to close the Submit Data Step Properties dialog box.

**16** To specify resources for the step, click the **Resources** tab. Click **Add** to add a resource's URL and Referer page.



## Modifying a Custom Request Step

A custom request represents a custom HTTP request for a URL, with any method supported by HTTP. A custom request step is contextless.

The properties that you can modify are the name of the step, method, URL, target frame, and body.

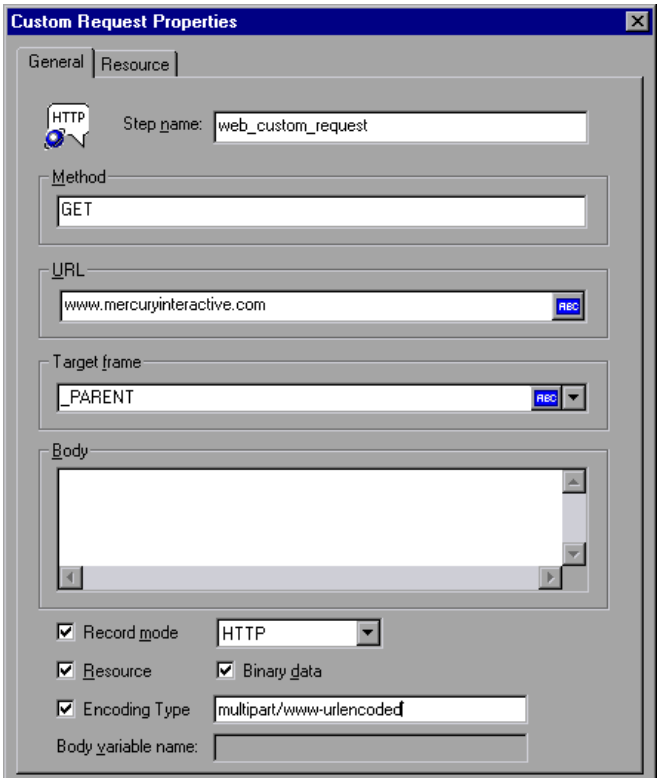
VuGen has a feature that lets you convert a custom request body string to C format. For example, if you copy an XML tree or a large amount of data into the Body area of the custom request, you can convert the strings to C format in order that it may be incorporated into the current function. VuGen inserts the necessary escape sequence characters and removes the line breaks in the string.

**To modify the properties of a custom request step:**



**1** In the tree view of the Vuser script, select the custom request step you want to edit. Custom request steps are shown using the **Custom Request** icon.

**2** Select **Properties** from the right-click menu. The Custom Request Properties dialog box opens.




**3** To change the step name, type a new name in the **Step name** box. The default name during recording is the last part of the URL.

**4** In the **Method** box, type any method supported by HTTP. For example, GET, POST or HEAD.

**5** In the **URL** box, type the URL being requested.

6 Select a **Target frame** from the list:

**_TOP:** replaces the whole page

**_BLANK:** opens a new window

**_PARENT**: replaces the parent of the last (changed) frame

**_SELF:** replaces the last (changed) frame

7 In the **Body** 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, refer to 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 640.

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 load testing or 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.

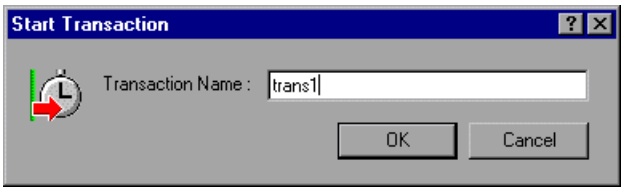
The properties that you can modify are the name of the transaction (start transaction and end transaction) and its status (end transaction only).

**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.



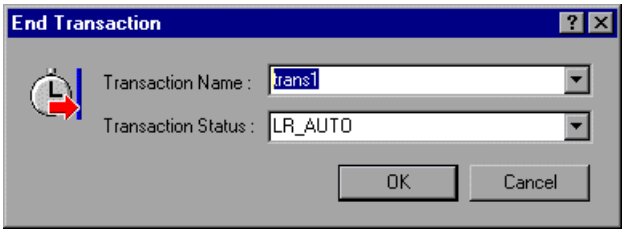

3 To change the transaction name, type a new name in the **Transaction Name** box, and click **OK**.

**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, refer to 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.

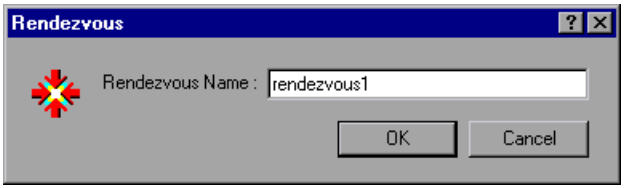
The property that you can modify is the name of the rendezvous point.

**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, you can instruct the Vuser to replay think time as recorded or ignore the recorded think time. For details, see Chapter 12, "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. Refer to 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.



*Image Check icon*

*Text Check icon*

2 Select **Properties** from the right-click menu. The appropriate Web check properties dialog box opens. This dialog box varies, depending on the type of check that you are modifying.

3 Type or select the properties required for the check. For details, see Chapter 44, "Verifying Web Pages Under Load."

4 Click **OK** to close the check properties dialog box.

# 46

# Setting 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 for another.

This chapter describes how to correlate statements during recording. It discusses:

- About Correlating Statements
- Understanding the Correlation Methods
- Using VuGen's Correlation Rules
- Setting Correlation Rules
- Testing Rules
- Setting the Correlation Recording Options

***The following information applies to Web and PeopleSoft Enterprise 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. Often, correlation will solve the problem by enabling you to use the results of one statement as input for another.

The dynamic data in an HTML page can be in the form of:

- a URL that changes each time you access the associated Web page
- 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, Vusers were unable to successfully replay the recorded script.

You can use correlated statements to resolve the difficulties in both of the above cases. Replace the dynamic data in the recorded script with one or more parameters. When the script runs, it assigns values to each of the parameters.

## Understanding the Correlation Methods

This chapter discusses automatic correlation using built-in or user-defined rules. To manually correlate statements, or to perform correlation for Wireless Vuser scripts, see "Performing Manual Correlation" on page 687.

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 Level" on page 575.

You can instruct VuGen to correlate the statements in your script either during or after recording. The recording-time solutions described in this chapter automatically correlate the statements in your script during recording time. You can also use VuGen's snapshot correlation to correlate scripts after recording. For more information on correlating after recording, see Chapter 47, "Correlating Vuser Scripts After Recording."

# Using VuGen's Correlation Rules

VuGen's correlation engine allows you to automatically correlate dynamic data during your recording session using one of the following mechanisms:

- Built-in Correlation
- User-Defined Rule Correlation

For additional information, see "Adding Match Criteria" on page 668 and "Advanced Correlation Rules" on page 668.

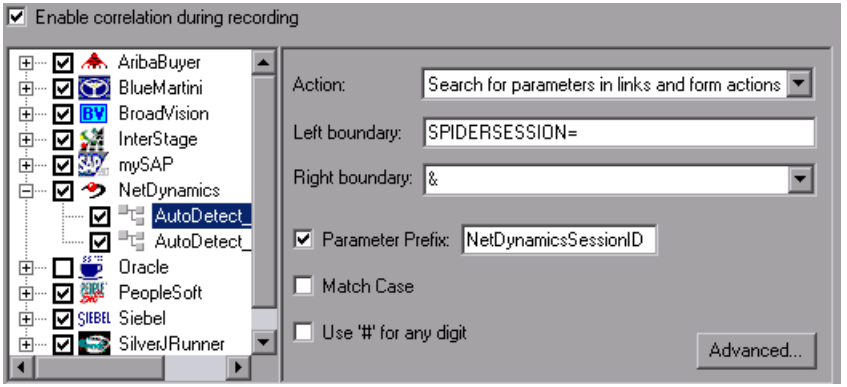
### Built-in Correlation

The Built-in correlation detects and correlates dynamic data for supported application servers. Most servers have clear syntax rules, or contexts, that they use when creating links and referrals.

For example, BroadVision servers create session IDs that are always placed between the same delimiters: "BV_SessionID=" on the left, and "&" on the right.

```
BV_SessionID=@@@@1303778278.0969956817@@@@&
```

If you are recording a session with a supported application server, you can use one of the existing rules built into VuGen. An application server may have more than one rule. You can enable or disable a specific rule by selecting or clearing the check box adjacent to the rule. VuGen displays the rule definitions in the right pane.




If you are recording a session on an unsupported application server whose context is not known, and you cannot determine any correlation rules, you can use VuGen's snapshot comparison method. This method guides you through the correlation procedure after you finish recording. For more information, see Chapter 47, "Correlating Vuser Scripts After Recording."

### User-Defined Rule Correlation

If your application has unique rules and you are able to determine them clearly, you can define new rules using the Recording Options.

User-defined rule correlation requires you to define correlation rules before you record a session. You create the correlation rules in the Recording Options dialog box. The rules include information such as the boundaries of the dynamic data you want to correlate and other specifications about the match such as binary, case matching, and the instance number.

You instruct VuGen where to search for the criteria:

- All Body Text
- Link/Form Actions
- Cookie Headers
- Form Field Value
- Insert Cookie Function

Note that by default, the maximum size of a string that you can save for a rule is 4096 characters. If necessary, you can modify this value by increasing the value of the **MaxParamLen** attribute in the **CorrelationSettings.xml** file, located in the Windows Installation directory.

### All Body Text

The **Search for Parameters in all of the Body Text** option instructs the recorder to search the entire body—not just links, form actions or cookies. It searches the text for a match using the borders that you specify.

### Link/Form Actions

The **Search for parameters in links and form actions** method instructs VuGen to search within links and 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 the right boundary alone is not sufficient, since it is not consistent—sometimes it is "@" and other times it is "&". In this case, you specify "&" as the alternate right boundary.

The left and right boundaries should uniquely identify the string. Do not include dynamic data in the boundaries. You can also specify **End of String** or **Newline Character** as a right boundary, available as options in the drop-down menu.

Note that for this option, the left and right boundaries must appear in the string that appears in the script—it is not sufficient for the boundaries to be returned by the server. This limitation does not apply to the other action types.

### Cookie Headers

The **Search for Parameters from Cookie Header** method is similar to the previous rule, except that the value is extracted from cookie text (exactly as it appears in the recording log) instead of from a link or form action.

In addition, the link/form action rule parameterizes only the part of URL that matches the rule boundaries. The cookie rule looks for the extracted value in links and action form fields and replaces it with a parameter automatically, without having to display the boundaries in the script.

### Form Field Value

The **Parameterize form field value** method instructs the recorder to save the named form field to a parameter. It creates a parameter and places it in the script before the form's action step. For this option, you need to specify the field name.

### Insert Cookie Function

The **Text to enter a web_reg_add_cookie function by** method inserts a **web_reg_add_cookie** function if it detects a certain string in the buffer. It only adds the function for those cookies with the specified prefix. For this option, you need to specify the search text and the cookie prefix.

## Adding Match Criteria

In addition to the above rules, you can further define the type of match for your correlation by specifying the following items for the string:

**Parameter Prefix:** Uses a prefix in all automatically generated parameters based on this rule. Prefixes prevent you from overwriting existing user parameters. In addition, prefixes allow you to recognize the parameter in your script. For example, in Siebel-Web, one of the built-in rules searches for Siebel_row_id prefix.

**Match Case:** Matches the case when looking for boundaries.

**Use "#" for any digit:** Replaces all digits with a hash sign. The hash signs serve as wildcard, allowing you to find text strings with any digit. For example, if you enable this option and specify **Mercury###** as the left boundary, **Mercury193** and **Mercury284** will be valid matches.

### Adding Comments

You can instruct VuGen to insert descriptive comments to the correlation steps within your script. To enable this option, select the **Add Comments to script** option.

## Advanced Correlation Rules

VuGen lets you specify the following advanced correlation rules:

**Always create new parameter:** Creates a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance. This option should be set if the Web server assigns a different value for each page. For example, NetDynamics servers may change the session ID from page to page to minimize fraud.

**Replace with parameter only for exact matches**: Replace the recorded value with a parameter only when the text between the boundaries exactly matches the found value (from the first snapshot). If there are additional characters either before or after the string, it will not replace the parameter.

For example, in a form submission, VuGen recorded the characters 1234 between the boundaries aaa and bbb, aaa1234bbb. In subsequent submissions of this form, VuGen only replaces the recorded value with a parameter if it finds the characters 1234, Name=1234. If another value is entered, even if it contains the first string, for example, Name=12345, VuGen will not replace the value with a parameter. Instead, it will use the value 12345.

**Reverse Search:** Searches for the left boundary from the end of the string backwards.

**Left boundary Instance:** The number of occurrence of the left boundary within the string (not the body) for it to be considered a match.

**Offset:** The offset of a sub-string of the found value to save to the parameter. The default is the beginning of the matched string. Note that you must specify a non-negative value.

**Length:** The length from its offset of a sub-string of the matched string to save to the parameter. If you disable this option, the default saves the string from the specified offset until the end of the match.

**Alternate Right Boundary:** An alternative criteria for the right boundary if the previously specified boundary is not found. You can specify text, **End of String**, or **Newline Character**.

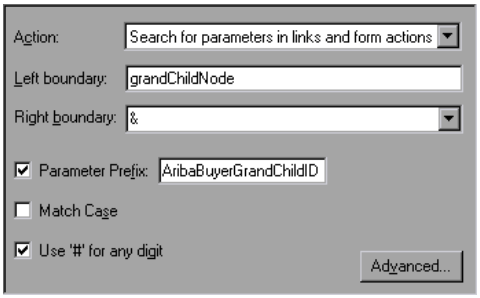
## Setting Correlation Rules

You can add, modify, or remove rules using the Correlation Recording options. Note that you can also edit rules that were created automatically for application server environments.

In addition to creating rules using the recording options before recording, you can create rules after recording. After running your script, you scan it for correlations (CTRL+F8). You select one of the correlation results, and create a rule based on its properties. For more information, see “Performing a Scan for Correlations” on page 683.

**To define correlation rules:**

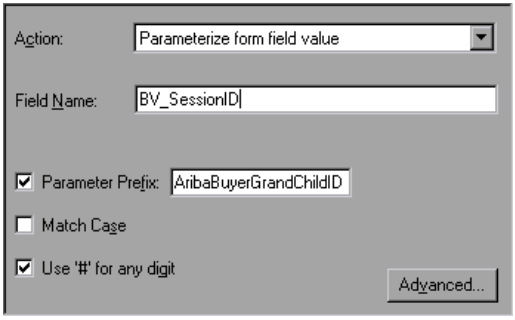
1 Click on an existing rule or click **New Rule** in the left pane. The Correlation Rules are displayed in the right pane.




2 Select a type of action: link or form action, cookie, all body, form field, or web_reg_add_cookie.

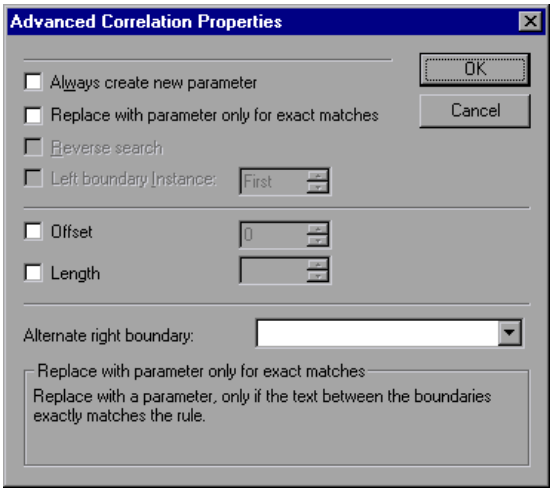
3 For the first three types, specify boundaries of the data in the **Left Boundary** and **Right Boundary** boxes.

4 For form field type actions, specify the field name.




5 Select the desired options: **Match Case** and/or **Parameter Prefix**. Specify a parameter prefix. To convert all digits to hash signs (#), select **Use # for any digit**.

**6** To set advanced rules, click **Advanced** in the Correlation node. The Advanced Correlation Properties dialog box opens.




- Select **Always create new parameter** to create a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance.
- Select **Replace with parameter only for exact matches** to replace a value with a parameter only when the text exactly matches the found value.
- Select **Reverse Search** to perform a backward search.
- Select the **Left Boundary Instance** box and specify the desired instance.
- Select **Offset** to specify an offset for the string within the match.
- Select **Length** to specify the length of the matched string to save to the parameter. This option may be used in conjunction with the **Offset** option.
- Specify another right boundary in the **Alternate right boundary** box or choose **End of String** or **NewLine Character** from the drop-down menu.

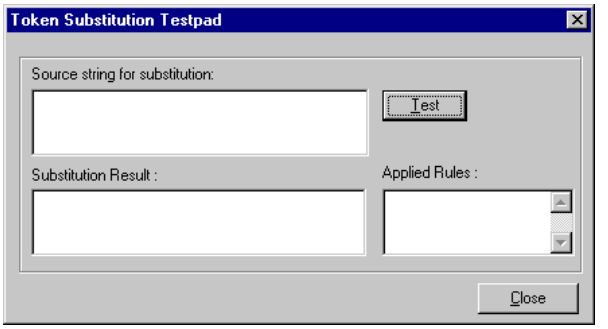
**7** Click **Test Rule** to test the rule you just defined. For information, see "Testing Rules" on page 672.

**8** Click **OK** to save the rules and close the dialog box.

# Testing Rules

This section applies to user-defined rules that you created for a server with a known context. After you define a new rule in the Correlation Rule dialog box, you can test it before recording your session by applying the rules to a sample string. You test the rules in the Token Substitution Testpad. To use the testpad:

1. Select a rule from the left pane and click **Test**. The Token Substitution Testpad dialog box opens.




2. Enter text in the **Source String for Substitution** box.
3. Click **Test**.

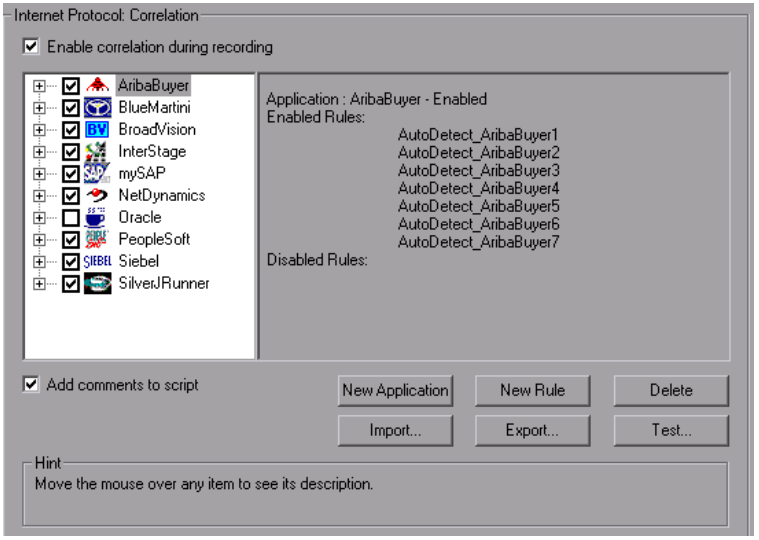
If substitution occurred, you will see the parameterized source text in the **Substitution Result** box and a list of rules that were applied to it in the **Applied Rules** box.

# Setting the Correlation Recording Options

To instruct VuGen to correlate your statements during recording, you set the Correlation recording options. You set these options after opening a Web Vuser script but before you begin recording the session.

**To set the correlation recording options:**

**1** After you create a script, but before you begin recording, select **Tools > Recording Options** and select the **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 669.

**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, VuGen issues a popup message.
8. To delete an application or rule, select it and click **Delete**. VuGen prompts you to confirm your choice 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**.

# 47

# 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:

- About Correlating with Snapshots
- Viewing the Correlation Results Tab
- Setting Up VuGen for Correlations
- Performing a Scan for Correlations
- Performing Manual Correlation
- Defining a Dynamic String's Boundaries

***The following information applies only to Web, Wireless, SAP-Web, and Siebel-Web Vuser scripts.***

## About Correlating with Snapshots

VuGen provides several correlation mechanisms for Web Vuser scripts. The automatic method discussed in Chapter 46, "Setting Correlation Rules for Web Vuser Scripts" detects dynamic values during recording and allows you to correlate them right away. If you disabled automatic correlation, or if the automatic method did not detect all of the differences, you can use VuGen's built-in correlation mechanism, described in this chapter, to find differences and correlate the values. You can also use this mechanism for scripts that were only partially correlated.

The correlation mechanism uses snapshots to track the results of script execution. Snapshots are graphical representations of Web pages. VuGen captures snapshots of the Web pages during record and replay. You compare the recorded snapshot to any of the replay snapshots to determine which values you need to correlate to successfully run the script. For more information about Record and Replay snapshots, see "Understanding Snapshots" on page 20.

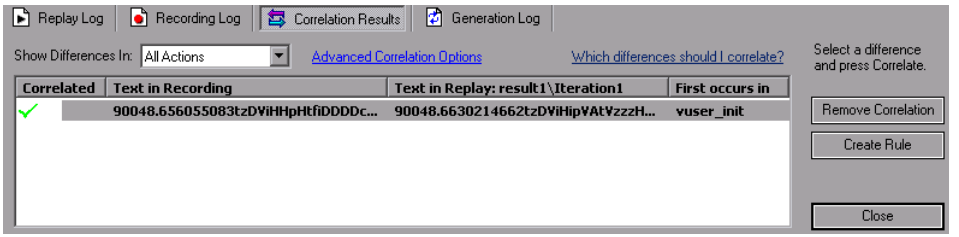
The Web correlation mechanism has a built-in comparison utility that allows you to view the text or binary differences between the snapshots. You can then correlate the differences one-by-one or all at once.

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 687.

# Viewing the Correlation Results Tab

The Correlation Results tab displays the differences between the Record and Replay snapshots.

When you instruct VuGen to scan the script for correlations, it opens the Output window and displays the differences between the recording and replayed snapshots in the **Correlation Results** tab.




You can display all the differences in the script or only those for the current step or action, by selecting the desired option from the **Show Differences In** list box.

Differences that were correlated are indicated by a check mark in the **Correlated** column. The next two columns, **Text in Recording**, **Text in Replay** show the text differences between the snapshots. The next column, **First occurs in**, indicates the Action in which the correlation was first detected.

After you detect the differences between the snapshots, you correlate them one at a time by selecting the correlation and clicking **Correlate**. VuGen also allows you to undo a specific correlation using the **Remove Correlation** button. If you expect one of the detected correlations to occur in subsequent recordings, you can create a new correlation rule. By creating rules, you enable VuGen to recognize differences during recording and automatically correlate them. For more information, see “Creating a Rule” on page 678.

When you correlate a value using the this mechanism, VuGen inserts a **web_reg_save_param** function and a comment into your script indicating that a correlation was done for the parameter. It also indicates the original value.

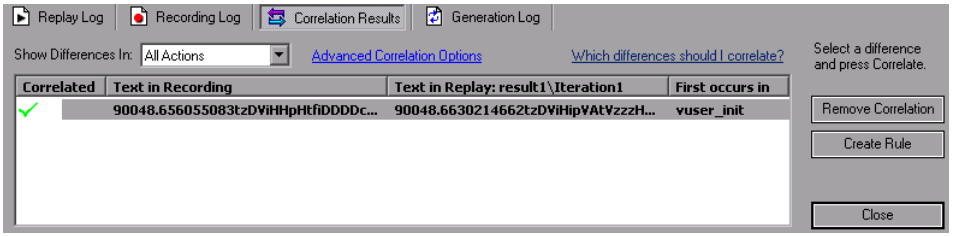
```
// [WCSPARAM WCSParam_Diff1 14 reserveFlights] Parameter
{WCSParam_Diff1} created by Correlation Studio
      web_reg_save_param( "WCSParam_Diff1", "LB= NAME=\"", "RB=\"",
"Ord=5", "Search=Body", "RelFrameId=1", LAST );
      web_submit_form("reservations.pl",
           "Snapshot=t4.inf",
           ITEMDATA,
           "Name=depart", "Value=Denver", ENDITEM,
           "Name=departDate", "Value=06/25/2004", ENDITEM,
           "Name=arrive", "Value=Los Angeles", ENDITEM,
           "Name=returnDate", "Value=06/26/2004", ENDITEM,
           "Name=numPassengers", "Value=1", ENDITEM,
           "Name=roundtrip", "Value=<OFF>", ENDITEM,
           "Name=seatPref", "Value=None", ENDITEM,
           "Name=seatType", "Value=Coach", ENDITEM,
           "Name=findFlights.x", "Value=44", ENDITEM,
           "Name=findFlights.y", "Value=12", ENDITEM,
           LAST);
      lr_think_time(12);
```

### Creating a Rule

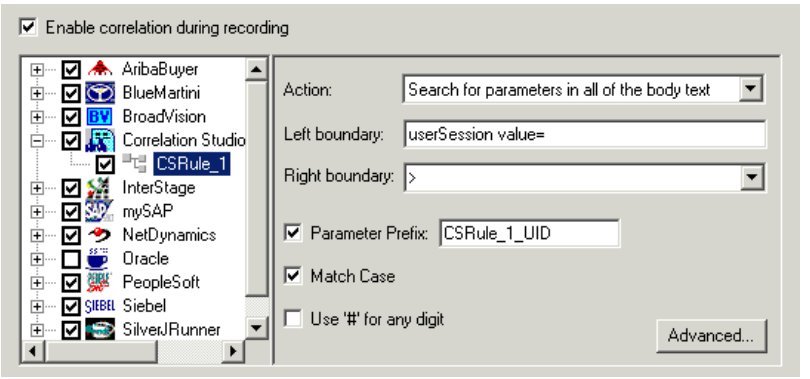
You can create a rule directly from the list of Correlated Results. Creating a rule, enables VuGen to recognize the difference during recording and automatically correlate it.

**To create a rule from one of the detected correlations:**

Select the correlation and click **Create Rule**. You can also create a rule by selecting a correlation and choosing **Create Correlation Rule** from the right-click menu.

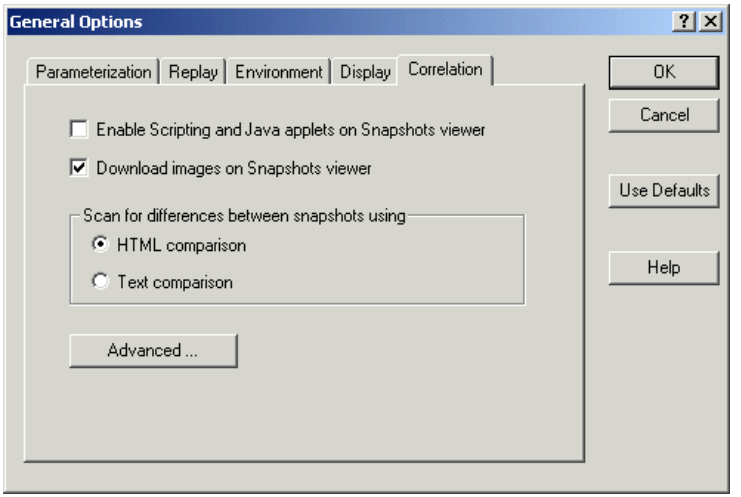



VuGen adds this rule to the list of **Correlation** rules. You can view this rule in the Recording Options Correlation node. In the following example, VuGen added the rule as CSRule_1.



# Setting Up VuGen for Correlations

You set the global Correlation setting under the General options. These options instruct the Vusers to save correlation information during replay, to be used at a later stage. You can specify the type of comparison to perform when comparing snapshots: HTML or text. In the Advanced options, you can indicate which characters should be treated as delimiters.




**Enable Scripting and Java applets on Snapshots viewer:** Allows VuGen to run applets and javascript in the snapshot window. This is disabled by default because it uses a lot of resources.

**Download images on Snapshots viewer:** Instructs VuGen to display graphics in the Snapshot view. If you find that the displaying of images in the viewer is very slow, you can disable this option. This option is enabled by default.

**Scan for differences between snapshots using:** Choose a comparison method:

- **HTML Comparison:** Only display the differences in HTML code.
- **Text Comparison:** Display all text, HTML, and binary differences.

---

**Note:** In most cases, 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.

---

**Advanced:** Opens the Advanced Correlation dialog box.

### Advanced Correlation dialog box

This dialog box lets you specify the characters to be treated as delimiters.

**Characters that should be treated as delimiters:** Specifies one or more non-standard delimiters.

**Additional Delimiters:** You can specify standard delimiters such as Carriage Return, New line and Tab characters. To change this setting, clear the checkbox next to the delimiter.

**Ignore differences shorter than … characters:** Allows you to specify a threshold for performing correlation. 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. The default value is 4 characters.

**Issue a warning for large correlations:** Issues a warning if you try to correlate a string whose size is 10 KB or larger.

### Setting the Correlation Preferences

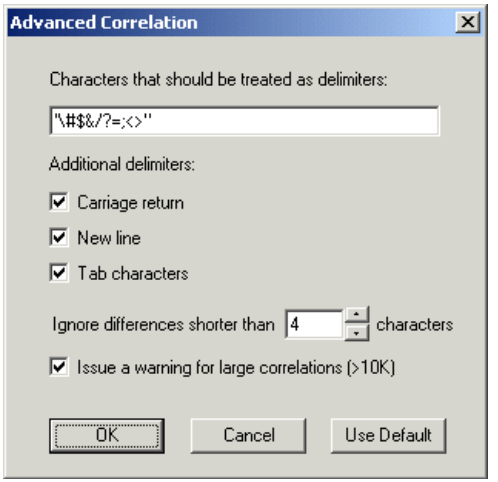
Before recording a session, you configure the correlation preferences.

**To set the correlation preferences:**

1. Choose **Options** > **General** and select the **Correlation** tab.
2. Select **Enable Scripting and Java applets on Snapshots viewer** to allow VuGen to run applets and javascript in the snapshot window.
3. To instruct VuGen to display graphics in the Snapshot view, select the **Download images on Snapshots viewer** option.

4 Choose the comparison method: **HTML comparison** or **Text Comparison** (for non-HTML elements only).

5 To set the delimiter characters, click **Advanced** to open the Advanced Correlation dialog box.




6 In the **Characters that should be treated as delimiters** box, specify all characters that are to be treated as delimiters.

7 Select the desired options in the **Additional delimiters** section, to specify one or more standard delimiters.

8 Specify a threshold for the correlation in the **Ignore differences shorter than** box. When VuGen compares the recorded 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.

9 To **issue a warning for large correlations**, select the option's check box.

10 Click **OK** to accept the Advanced Correlation settings and close the dialog box.

11 Click **OK** in the General Options dialog box to accept the Correlation setting and close the dialog box.

# Performing a Scan for Correlations

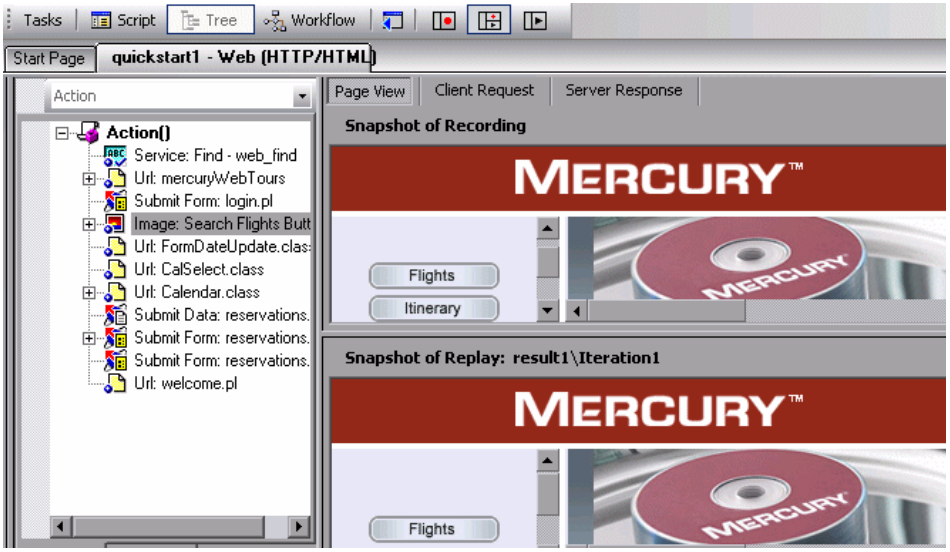
You can use VuGen's snapshot window to determine which values within your script are dynamic and require correlation. The following section describes how to automatically scan the script for differences and use VuGen to perform the necessary correlations.

**To scan your script for correlations:**

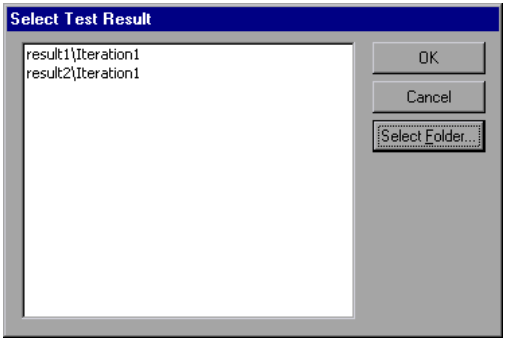
1 Open a script and view it in Tree view (**View > Tree View).** Display the snapshots (**View > Snapshot > View Snapshot).**

2 Select a script step in the Tree view from the left pane. A snapshot opens in the right pane.

3 To display both the recording snapshot and the first replay snapshot, click **View > Snapshot > Recorded and Replayed**.

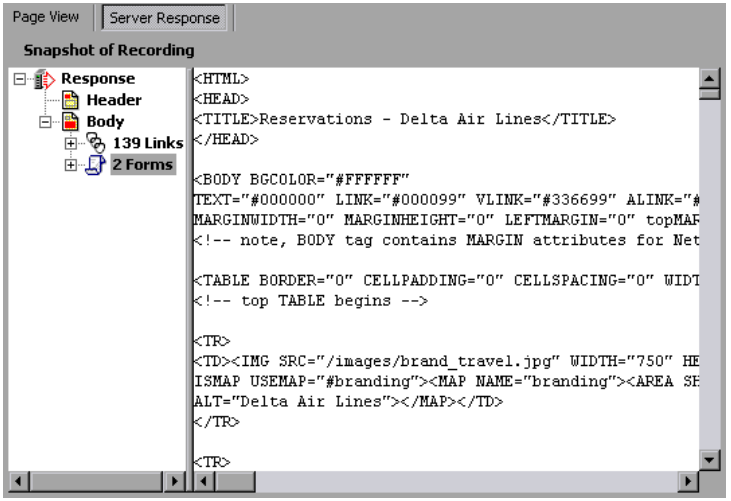


**4** To use a snapshot other than the first, click **View** > **Snapshot** > **Select Iteration.** A dialog box opens, displaying the folders that contain snapshot files. These are usually the **result** and **Iteration** folders below the script's folder.



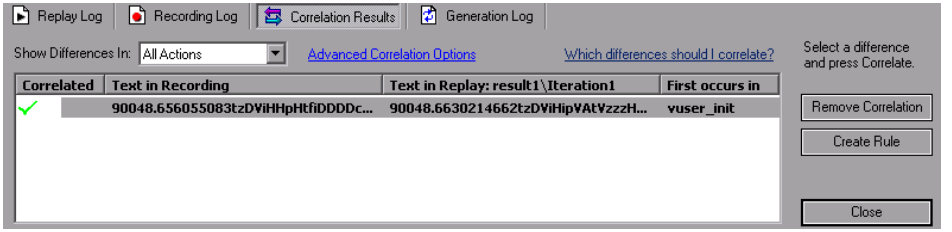

**5** To select a snapshot file in a folder other than the one in the subfolders of the script, click **Select Folder**. Browse to the desired location, and click **OK**.

**6** To view the HTML code, click the **Server Response** tab. Expand the Body branch.




To return to the page view, click the **Page View** tab.

**7** Choose **Vuser** > **Scan for Correlations** or click the **Find Correlations** button. VuGen scans the script for dynamic values that need to be correlated and displays them in the Correlation Results tab.




**8** View all differences or choose a filter method in the **Show Differences In** list box. The options are **All Actions**, **Current Action**, or **Current Step Only**.

### Determining the Differences to Correlate

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

The following strings most probably require correlation:

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

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

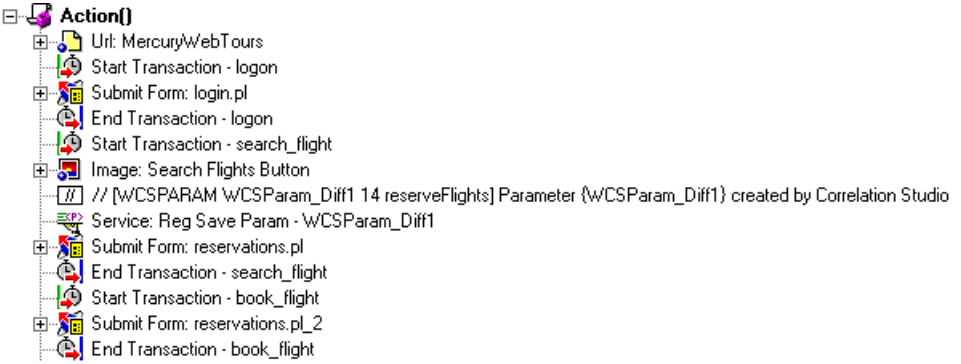
You should also correlate differences in which some of the recorded and replayed strings are identical, but others differ. For example, SessionID strings with identical prefixes and suffixes, but different characters in between, should be correlated.

Once you determine that a difference needs to be correlated, you instruct VuGen to correlate it.

**To correlate the differences:**

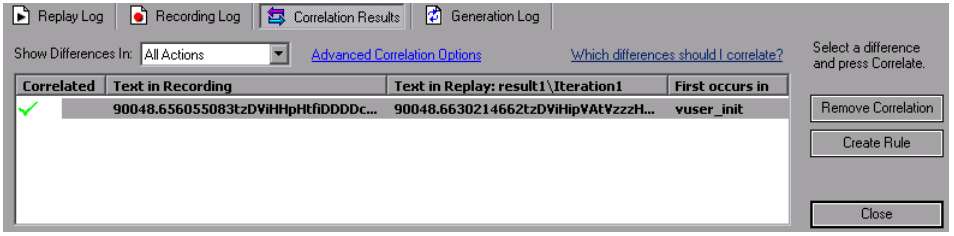
1 View the differences in the Correlation tab, and select the one you want to correlate. It is recommended that you correlate only one difference at a time.

2 Click **Correlate**. VuGen places a green check mark next to differences that were correlated and inserts a **web_reg_save_param** function into the script.

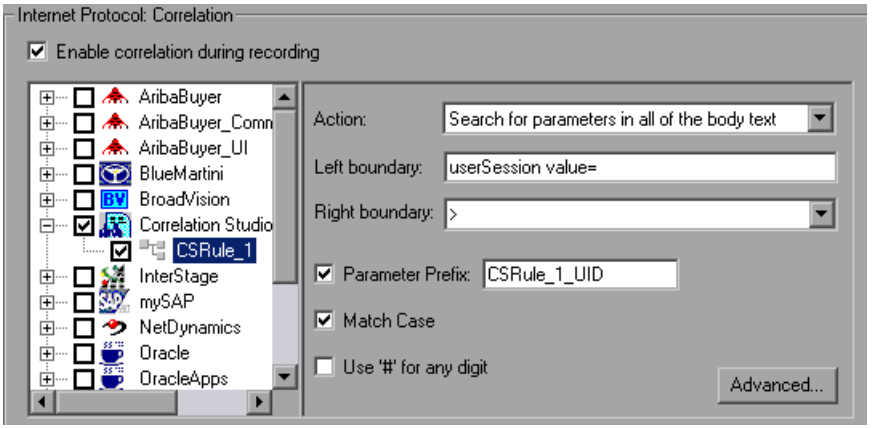



Repeat this step for all differences you want to correlate.

3 To create a rule from one of the detected correlations, select the correlation and click **Create Rule**. This is also available from the right-click menu. VuGen issues a message confirming that your rule was created.



To view this rule, open the Recording Options (CTRL +F7) and select the **Correlation** node. Expand the **Correlation Studio** entry and select your rule.




4 To undo a correlation, select the difference and click **Remove Correlation.**

5 Choose **File** > **Save** to save the changes to your script.

## Performing Manual Correlation

For Web Vusers, VuGen's automatic or rule-based correlation usually correlates the scripts dynamic functions so that you can run the script successfully. You can also perform correlation after the recording session using VuGen's snapshot comparison.

For Wireless Vusers and other Vuser scripts for which automatic correlation did not apply, VuGen also allows you to manually correlate your scripts. You manually correlate a script by adding the code correlation functions. The function that allows you to dynamically save data to a parameter is **web_reg_save_param.**

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

The function's syntax is as follows:

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

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

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

| | |
|---|---|
| Search | The scope of the search—where to search for the delimited data. The possible values are Headers (search only the headers), Body (search only Body data, not headers), or ALL (search Body and headers). The default value is ALL. |
| ORD | This optional parameter indicates the ordinal or occurrence number of the match. The default ordinal is 1. If you specify "All," it saves the parameter values in an array. |
| SaveOffset | The offset of a sub-string of the found value, to save to the parameter. The default is 0. The offset value must be non-negative. |
| Savelen | The length of a sub-string of the found value, from the specified offset, to save to the parameter. The default is -1, indicating until the end of the string. |
| Convert | The conversion method to apply to the data:<br>HTML_TO_URL: convert HTML-encoded data to a URL-encoded data format<br>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 692.
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 690 or the *Online Function Reference* (**Help > Function Reference**).

## Replacing Dynamic Data with a Parameter

Identify the actual dynamic data in the recorded statement, then search the entire script for the dynamic data and replace it with a parameter. Give the parameter any name and enclose it with braces: {param_name}. You can include a maximum of 64 parameters per script.

**To replace dynamic data with a parameter:**

Select **Edit** > **Replace** from the VuGen main window to display the Search and Replace dialog box. Search the entire script for the dynamic data and replace it with a parameter.

## Adding a Correlation Function

You insert the **web_reg_save_param** statement to save dynamic data in a script. This function tells VuGen to create a parameter that saves the run-time value of the dynamic data during replay.

When you run the script, the **web_reg_save_param** function scans the subsequent HTML page that is accessed. It searches for an occurrence of the left boundary, followed by any string, followed by the right boundary. When such an occurrence is found, VuGen assigns the string between the left and right boundaries to the parameter named in the function's argument. After finding the specified number of occurrences, **web_reg_save_param** does not search any more HTML pages. The Vuser continues with the next step in the script.

## Sample Correlation for Web Vusers

Suppose the script contains a dynamic session ID:

```
web_url("FirstTimeVisitors",
      "URL=/exec/obidos/subst/help/first-time-visitors.html/002-8481703-
        4784428>Buy books for a penny ",
      "TargetFrame=",
      "RecContentType=text/html",
      "SupportFrames=0",
       LAST);
```

You insert a **web_reg_save_param** statement before the above statement:

```
web_req_save_param ("user_access_number", "NOTFOUND=ERROR",
"LB=first-time-visitors.html/","RB=>Buy books for a penny", "ORD=6",
LAST);
```

After implementing correlated statements, the modified script looks like this, where **user_access_number** is the name of the parameter representing the dynamic data.

```
web_url("FirstTImeVisitors",
      "URL=/exec/obidos/subst/help/first-time-"
        "visitors.html/{user_access_number}Buy books for a penny ",
      "TargetFrame=",
      "RecContentType=text/html",
      "SupportFrames=0",
       LAST);
```

---

**Note:** Each correlation function retrieves dynamic data once, for the subsequent HTTP request. If another HTTP request at a later point in the script generates new dynamic data, you must insert another correlation function.

---

## Sample Correlation for Wireless Vusers

Suppose your script contains a dynamic session ID for a WAP connection:

```
web_url("login.po;sk=IuZSuuRlHUMnpF-wpK8PzEpy(1YOSBSMy)",
      "URL=http://room33.com/portal/login.po;sk=IuZSuuRlHUMnpF-
        wpK8PzEpy(1YOSBSMy)",
      "Resource=0",
      "RecContentType=text/vnd.wap.wml",
      "Mode=HTML",
      LAST);
```

You insert a **web_reg_save_param** statement before the above statement and replace the dynamic value with the parameter. In the following example, the **web_reg_save_param** functions saves the login ID string to a variable called SK. It saves binary data, denoted by the **RB/BIN** attribute, and sets the left boundary as "sk=".

```
web_reg_save_param(
   "SK",
"LB=sk=",
"RB/BIN=#login\\x00\\x01\\x03",
"Ord=1",
 LAST);


web_url("login.po;sk={SK}",
     "URL=http://room33.com/portal/login.po;sk={SK}",
     "Resource=0",
     "RecContentType=text/vnd.wap.wml",
     "Mode=HTML",
     LAST);
```

## Defining a Dynamic String's Boundaries

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

- Always analyze the location of the dynamic data within the HTML code itself, and not in the recorded script.
- Identify the string that is immediately to the left of the dynamic data. This string defines the left boundary of the dynamic data.
- Identify the string that is immediately to the right of the dynamic data. This string defines the right boundary of the dynamic data.

➤ **web_reg_save_param** looks for the characters between (but not including) the specified boundaries and saves the information beginning one byte after the left boundary and ending one byte before the right boundary. **web_reg_save_param** does not support embedded boundary characters. For example, if the input buffer is {a{b{c} and "{" is specified as a left boundary, and "}" as a right boundary, the first instance is c and there are no further instances—it found the right and left boundaries but it does not allow embedded boundaries, so "c" is the only valid match.

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

```
web_set_max_html_param_len(“1024”);
```

# 48

# Testing XML Pages

VuGen's Web Vusers support Web pages containing XML code.

This chapter describes:

- About Testing XML Pages
- 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.

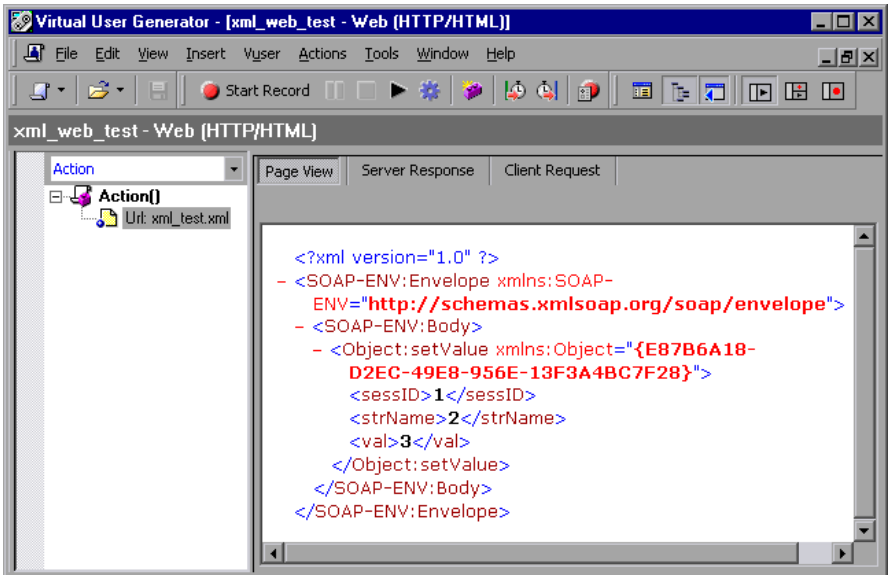
The XML code can appear in the script as a regular URL step or as a custom request. VuGen detects the HTML and allows you to view each document type definition (DTD), its entities, and its attributes. VuGen can interpret the XML when the MIME type displayed in the **RecContentType** attribute or the MIME type returned by the server during replay, ends with **xml**, such as **application/xml** or **text/xml**. The DTD is color coded, allowing you to identify each one of the elements. You can also expand and collapse the tree view of the DTD.

When you expand the DTD, you can parameterize the attribute values. You can also save the values in order to perform correlation using the standard correlation functions. For more information about the correlation functions, refer to the *Online Function Reference* (**Help > Function Reference**).

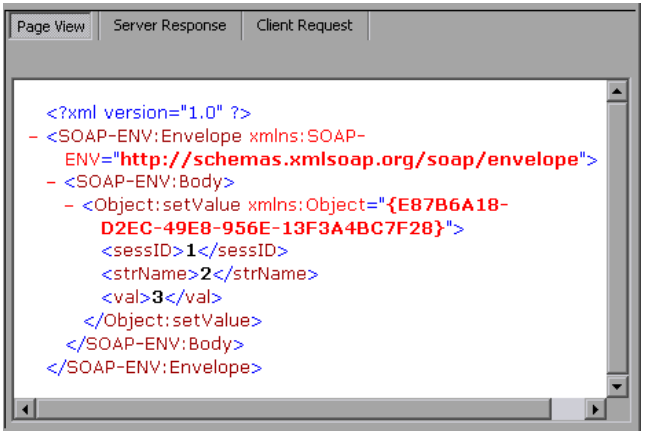
**Note:** VuGen cannot display a DTD with **XML islands**, segments of XML embedded inside an HTML page. VuGen only displays pages that are entirely XML.

## Viewing XML as URL Steps

One way to test a page with XML code, is to record it with VuGen. You record the XML pages as you would record a standard Web page. VuGen records the DTD and all of the XML elements. It does not create a snapshot for the XML page. Instead, for each XML step it displays the XML code in the snapshot frame under the Server Response tab.

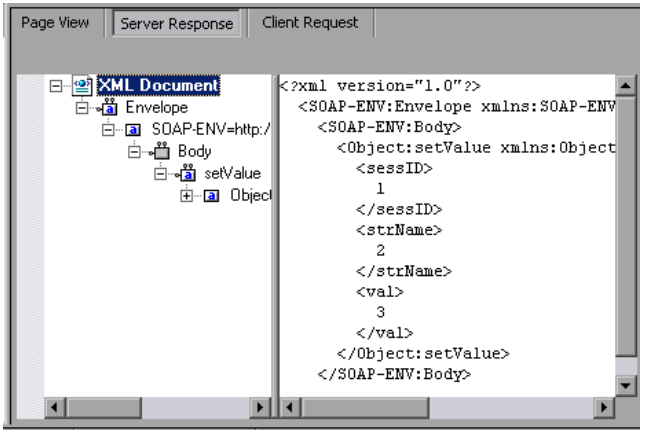


VuGen creates a color-coded expandable hierarchy of the DTD in the snapshot frame. Click on the "+" to expand an item, and click on the "-" to collapse it. VuGen displays all XML tags in brown, and values in black.

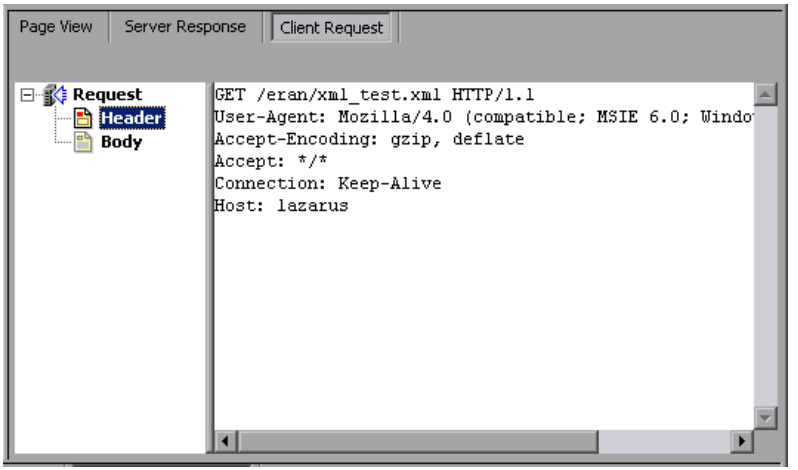



To replace any of the constant values with a parameter, select a value, perform a right-click, and select **Replace with a Parameter**. Follow the standard procedure for parameterization. For more information, see Chapter 8, “Working with VuGen Parameters.”

You can also view the Server response and Client request for the XML page by clicking the appropriate tab. The following example shows the Server response of an XML page. Note that you can expand and collapse all branches of the XML tree.



The following example shows the Client Request for the header of an XML page:



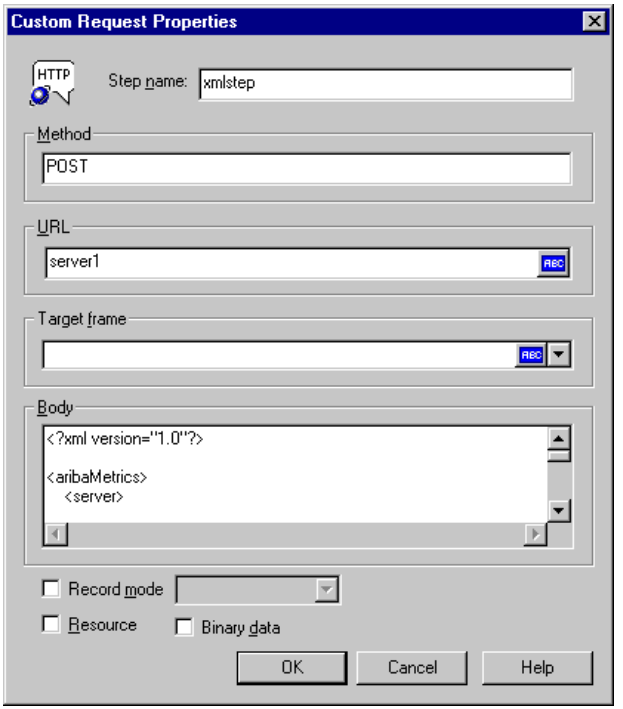

## Inserting XML as a Custom Request

You can also test your XML pages by inserting the XML code as a custom request. In this mode, the **Custom Request** properties box displays the elements of the DTD in either text or XML format.

**To add XML code as a Custom Request:**

1. View the script in tree view mode, place the cursor at the desired location, and 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 45, "Modifying Web and Wireless Vuser Scripts."

**6** Click **OK**. VuGen places the custom request step into your script.

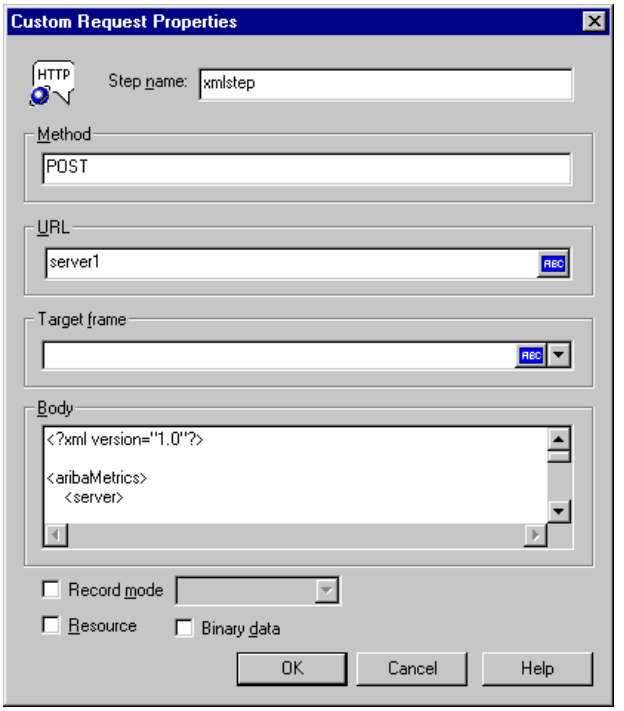
# Viewing XML Custom Request Steps

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.

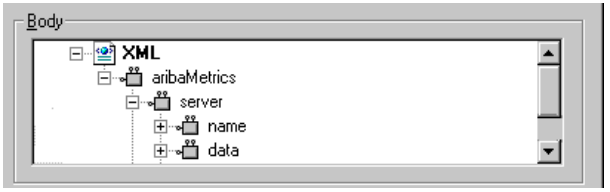
**To view the XML code of a custom request step:**

1 View the script in tree view mode, and select the desired step.

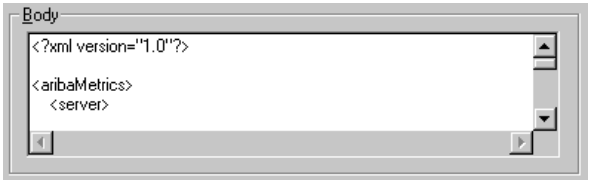
2 Choose **Properties** from the right-click menu. The Custom Request Properties dialog box opens.



The bottom section of the dialog box displays the XML code. If the **RecContentType** attribute is set to **text/xml**, by default VuGen displays the code in an XML format hierarchy. In this mode, the XML code is not editable.

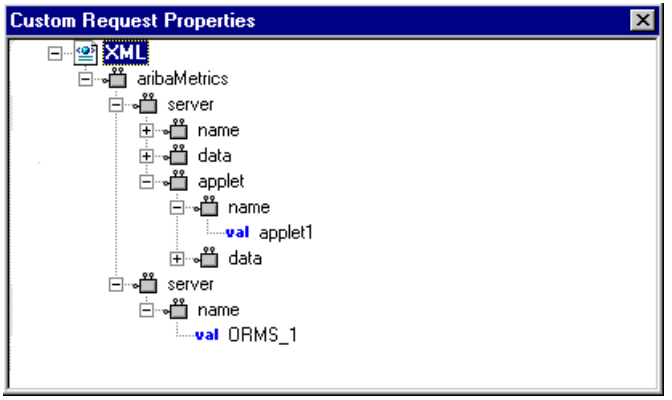



If the **RecContentType** attribute is set to any type other than **text/xml**, VuGen displays the code in plain text format. In this mode, the XML code is editable.




3 To switch between the text and XML views, 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.



# 49

# 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:

- About Using Reports to Debug Vuser Scripts
- Understanding the Results Summary Report
- Filtering Report Information
- Searching Your Results
- Managing Execution Results

---

**Note:** To enable all the VuGen Web report features, it is recommended that you work with Microsoft Internet Explorer 5.0 or later.

---

***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 14, "Running Vuser Scripts in Standalone Mode."

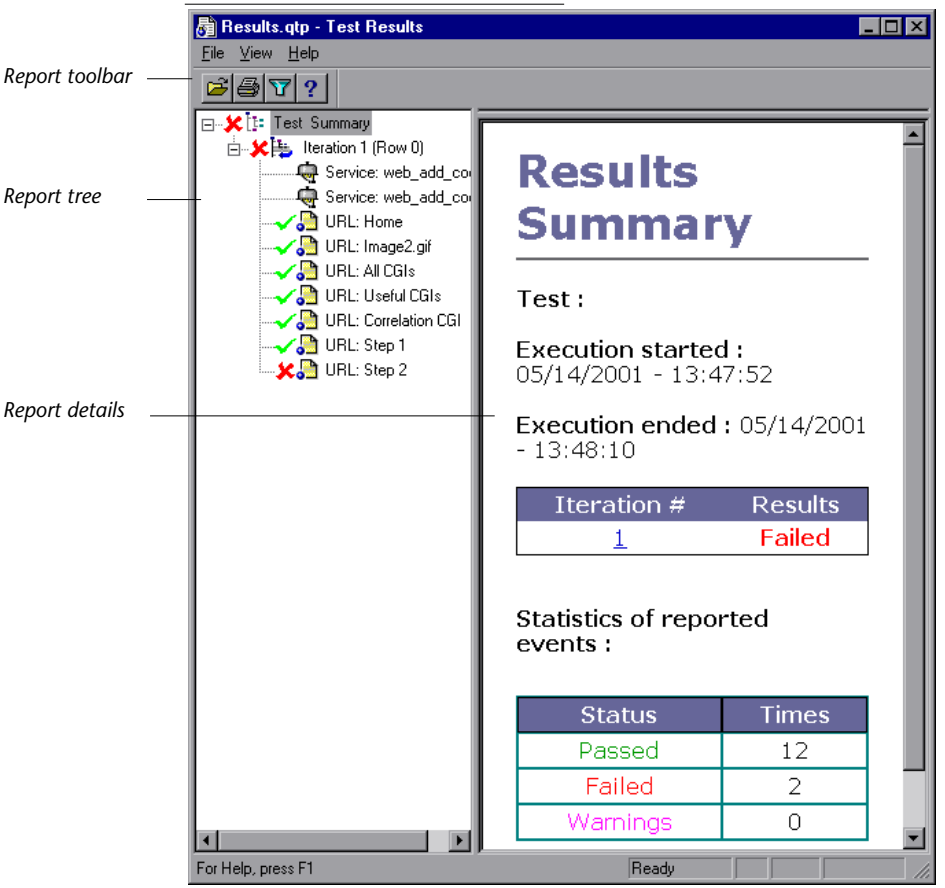
After you run a Vuser script using VuGen, you view the Results Summary report.

VuGen generates the results in VuGen report format—with a **.qtp** extension—and you view the results in the Virtual User Generator Report window. This is the recommended option because VuGen's Report window provides you with a more sophisticated interface and additional features.

You set the Display options (**Tools** > **General Options**) to specify whether or not VuGen should generate a Results Summary report, and if so, whether the report opens automatically after script execution. For details on setting the Display options, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

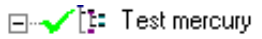
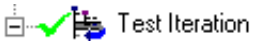
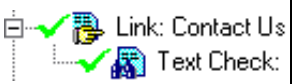
# Understanding the Results Summary Report

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 this branch… | To view the following details: |
|---|---|
| **Test Name**   | the overall results summary of the script execution |
| **Test Iteration**   | the execution summary for a specific iteration |
| **Test Step or 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**.

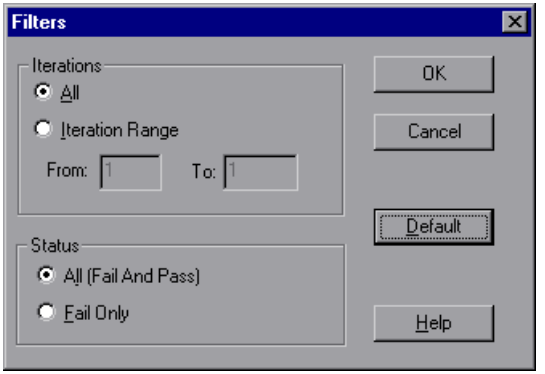
# Filtering Report Information

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.

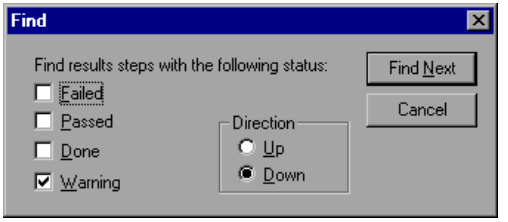
## Searching Your Results

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 215 of Chapter 14, “Running Vuser Scripts in Standalone 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.

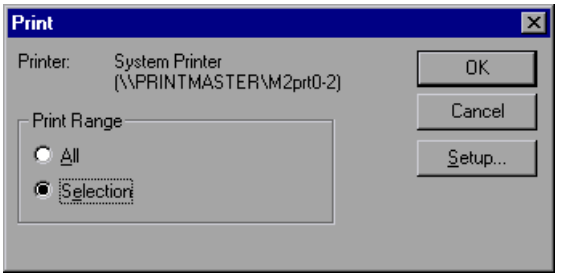
## Printing Report Results

You can print a Test Results Summary report.

**To print a Test Summary report:**



1 Select **File** > **Print**, or click the **Print** button on the **Report** toolbar. The Print dialog box opens.




2 Select a range from the **Print Range** box:

**All**—prints the entire report. This includes the Web page for each step in an iteration.

**Selection**—prints the selected branch in the **Report** tree.

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 Test Summary Report

To close a Test Summary report, select **File** > **Exit**. The Test Results window closes.

# 50

# 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 and Compatibility 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 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 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, prior to VuGen 7.0, **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, 7.0 and higher, **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 just 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 your application's **bin** directory. 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:** In Sockets mode, VuGen supports 128-bit encryption, independent of the browser version. In WinINet mode, VuGen supports the same encryption as the browser on the Load Generator machine. Both Netscape Communicator (4.5 and higher) and Internet Explorer (5.0 and higher) support 128-bit encryption.

**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, 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. Sometimes, however, VuGen is unable to 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. Sometimes, however, VuGen is unable to 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.<br><br>Workaround: After recording a Vuser script, insert correlation statements to for all **web_add_cookie** statements that set session cookies. Do not delete **web_add_cookie** calls 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.<br><br>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 about 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 stops JavaScript errors from appearing 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**: Can I display a run-time viewer when I run a Vuser from the Controller?

**Answer:** Yes. To display a run-time view from the Controller, begin running the Vuser, and choose **Vuser** > **Show Vuser**.

**Question 20**: What should I install on my load generator so that I will be able to display a run-time viewer?

**Answer:** Since the run-time viewer uses an Internet Explorer ActiveX control, you must have Microsoft Internet Explorer 4.0 or higher installed in order to use the run-time viewer.

**Question 21**: 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 22**: 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 23**: 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.

VuGen 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 most straight-forward way to install WinInet.dll version 4.0 is to install Internet Explorer 4.0 or higher.

**Question 24**: 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:

- SOCKS proxy record/replay
- Kerberos authentication

**Question 25**: Must I use a standard browser—such as Netscape or Internet Explorer—when I record?

**Answer:** You can use the browser of your choice when you record a Web Vuser script. You can also use a non-browser application that generates HTTP(S) requests. The only requirement of the application is that for single Web protocol scripts, you must be able to set the proxy settings to localhost:7777 to allow the recording of HTTP(S) requests. This is not required for multi-protocol scripts.

**Question 26**: How do I record a non-standard HTTP(S) application?

**Answer:** For a multi-protocol script, locate the application in the Start Recording dialog box. Make sure to enter the relevant command line parameters. For a single protocol Vuser script, perform the following procedure:

1. Choose **Tools** > **Recording Options** and click the **Browser** node. Select **Manually launch an application**.
2. Click the **Start Recording** button. VuGen prompts you for the proxy settings required for the recorded application. Note the host and port name.
3. Edit the proxy settings in the application being recorded. Make note of the original settings in order to restore them after the recording.
4. Click the **Start Recording** button and begin recording the session.
5. Close the application when you are finished recording and restore the original proxy settings (failure to do so may prevent it from working).

**Question 27**: Does VuGen ever modify any of the proxy settings in my recording browser?

**Answer:** Yes, for single protocol scripts only. 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 28**: 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 27 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. This only applies to single protocol scripts.

**Question 29**: 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** node. Select the **Obtain the proxy setting from the recording browser** option.

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 30**: 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 31**: 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 **web_set_sockets_options** in the *Online Function Reference*.

## Configuration and Compatibility Issues

**Question 32**: I performed a snapshot comparison and the results were very inaccurate.

**Answer:** Choose **Options** > **General** to open the General Options dialog box, and select the Correlation tab. In the **Scan for differences between snapshots using** section, make sure to choose the **HTML Comparison** option—not Text. Text comparison only applies to non-HTML snapshots.

**Question 33**: Can I replay a recorded script on a UNIX system?

**Answer:** Yes, replay is supported on UNIX platforms.

# 51

# Planning Web Service Tests

You use VuGen to create a script by recording a Web Service session or by importing a WSDL. When you run the script or LoadRunner Tuning Module session step, Vusers emulate real users communicating with the Web Service.

This chapter describes:

- About Planning Web Service Tests
- Implementing a Web Service
- Challenges in Web Services Testing
- Choosing a Web Services Script Type
- Performing a Load Test
- Client Emulation

# About Planning Web Service Tests

The term **Web Services** describes self-contained applications that can run universally across the Internet. Using Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP), they serve as building blocks for the rapid development and deployment of new applications. Since all communication is in XML, Web Services are not limited to a specific operating system or programming language. Web Services, therefore, allow applications from various sources to communicate with each other without extra coding and without intimate knowledge of each other's IT systems behind the firewall.

Unlike traditional client/server models, such as a Web page/Web server system, Web Services do not provide a user interface. Instead, Web Services share business logic, data and processes through a programmatic interface across a network. Developers can integrate a Web Service within an existing a user interface (such as a browser or an executable program) to offer specific functionality to users.

Web Services are considered B2B (Business-to-Business) applications, and as a result, they have strict response requirements. The fact that Web Services B2B applications communicate without a user interface, causes them to have better response times than those with user interfaces, such as browser-based applications. These factors make performance and load testing indispensable before the launching of Web Service application.

The key to building an effective testing plan, however, depends on your understanding of your system configuration and requirements. While VuGen can help you simplify the testing process, you should have an in-depth understanding of your system's requirements—its capacity and scalability. These factors are crucial to planning, designing, and performing a full-scale load test.

The following sections describe the challenges in Web Service testing and the ways to overcome them using Mercury tools.

# Implementing a Web Service

There are four primary steps in using a Web Service:

- Search for an appropriate Web Service
- Locate the URL of the Web Service
- Determine the Communication Protocol and Syntax
- Communicate

### Searching for a Web Service

The first step is to find an appropriate Web Service. Large software vendors have formed a universal directory for Web Services called the Universal Description, Discovery, and Integration (UDDI) services.

### Establishing the Service's URL

In order to use the desired service, you need to access it through its URL. For example, http://myservice.com/WebServices/MyService.asmx.

### Determining the Means of Communication

After finding the Web Service, you retrieve information about how you want to communicate with the service. This information is usually stored in a Web Services Description Language (WSDL) document. The WSDL document uses XML to define Web Services as collections of network endpoints, or ports, that characterize the physical network. VuGen allows you to import WSDL documents, generating readable code within the script.

### Communicating

Once you establish the means of communication and the properties of the network service, you can communicate. To communicate, you use structured XML documents. Using the Web Services protocol, VuGen records all of the communication and generates readable functions.

### Understanding a WSDL document

Each WSDL document defines the following elements for a Web Service:

- **Types**: a container for data type definitions using some type system (such as XSD).
- **Message**: a definition of the data being communicated.
- **Operation**: a description of an action supported by the service.
- **Port Type**: a set of operations supported by one or more endpoints.
- **Binding**: a protocol and data format specification for a particular port type, such as SOAP 1.1, HTTP GET/POST, and MIME.
- **Port**: a single endpoint defined as a combination of a binding and a network address.
- **Service**: a collection of related endpoints.

In the WSDL document, some of the elements are abstract, implying that they are not specific to a particular network service and may be reused. Examples of abstract elements are **messages** and **operations**.

Other elements are concrete, implying that they have values that are specific to each communication. An example of a concrete element is **port**.

WSDL uses the **binding** element to attach a specific protocol, data format, or structure to an abstract message, operation, or endpoint. Since they are abstract definitions, the message, operation and endpoint elements can be reused for other communications. You define a **port** by associating a network address with a binding. A collection of ports define a **service**.

The following example shows the WSDL definition of a service providing stock quotes. The service supports a single operation called GetLastTradePrice, which is deployed using the SOAP 1.1 protocol over HTTP. The request takes a ticker symbol of type string, and returns the price as a float.

This example uses a fixed XML format instead of the SOAP encoding.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
```

This section defines the **type** elements of the data:

```
 <types>
     <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
       <element name="TradePriceRequest">
         <complexType>
            <all>
               <element name="tickerSymbol" type="string"/>
            </all>
         </complexType>
       </element>
       <element name="TradePrice">
         <complexType>
            <all>
               <element name="price" type="float"/>
            </all>
         </complexType>
       </element>
     </schema>
   </types>
```

The following section defines several **message** elements:

```
<message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>

<message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
   </message>
```

This section defines a **portType** element, associated with an operation:

```
<portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
      </operation>
   </portType>
```

The following section defines the **binding** element The binding attaches the GetLastTaradePrice operation to the SOAP over HTTP protocol:

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePort-
Type">
      <soap:binding style="document" transport="http://schemas.xml-
soap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation soapAction="http://example.com/GetLastTrade-
Price"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
            <soap:body use="literal"/>
         </output>
      </operation>
   </binding>
```

The following section defines the final element, **service**. A service consists of one or more ports:

```
 <service name="StockQuoteService">
      <documentation>My first service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
      </port>
   </service>

</definitions>
```

## Challenges in Web Services Testing

Testing Web Services presents several challenges:

**Non-standard UI:** One of the biggest challenges in testing complex Web Services, is testing the client. Web Services do not display a user interface that can be tested. Often, the Web Services is an application or server—not a Web browser or thin client application using the traditional client-server approach. In addition, the client application might be driven by another application, which also lacks a user interface.

**Isolating the business process:** It is difficult to isolate the actual business process from the other Web Services traffic. If you record the client application, your script may contain unnecessary calls to methods that are not related to the business process you want to test.

**Complex logic:** Web Services often use complex logic, with elements whose values change during each test run. These values must be correlated and saved to parameters, in order to be used in subsequent calls to the Web Services methods.

**Timing and Order:** There are instances where the client make asynchronous calls, not related to the business process. Calls may have been issued asynchronously by the application or issued by another business process that was initiated by another application. For example, the Web Service may initiate a backup procedure every hour, independent of the current business process. In addition, the order of calls to Web Service methods may change from run to run.

**Performance Analysis:** Performance analysis is also a challenging task in Web Services testing. After you finish running a load scenario, you need to try to determine where the problem occurred. Since the Web Services system can involve several servers, possibly on a distributed system, it may be difficult to pinpoint the problematic component.

You can overcome most of these issues using the Web Services Vuser, adding think time, and applying correlation. For more information on selecting the best method for creating a script, see "Choosing a Web Services Script Type" on page 730.

Using Mercury's LoadRunner Controller and Analysis tools, you can overcome the performance analysis issues by setting monitors and performing transaction breakdown throughout the distributed system.

## Choosing a Web Services Script Type

You can create a Web Services Script in one of the following ways:

- Recording a Web Services Script
- Scanning a WSDL Document
- IDE Integration

### Recording a Web Services Script

You can record a Web Service application using VuGen's built-in recorder.

When you record an application, you can record it with or without a WSDL file. When you import a WSDL into your recording, it allows VuGen to create hi- level code, by parsing the recorded SOAP traffic and extracting the calls to the Web Service.

The advantages of recording a client session is it's simplicity. While recording, you perform actions to create a script that emulates a specific business process. You can create transactions and monitor its traffic, its timing, order of calls, and you can monitor the data that passes between peers. In addition, the scripts can be used for both functional and load testing.

A drawback of this technique is the fact that it captures all of the HTTP from the application, even packets that do not relate to the specific business process that you are trying to record. Therefore, additional filtering of the recorded script may be required.

### Scanning a WSDL Document

This approach is ideal for systems where the script needs to be independent and interoperable. VuGen uses client emulation that sends headers (HTTP) and namespace prefixes (XML) to accurately emulate the toolkit. Another advantage is that this script can be used for both functional and load testing.

A drawback of the Scan WSDL method is that the WSDL does not have any timing information, nor does it know the call order of the methods. To overcome the timing issues, you can insert think time delays.

Another disadvantage of this approach is the need to manually enter parameter values. By manually entering argument values, especially for complex Web Service methods, the likelihood of inserting erroneous data increases, resulting in faulty test results. Note that you can facilitate this by exporting the XML file for each method after manually entering the values. Then, for future tests, you can use the XML which contains the values that you entered earlier. For more information, see Chapter 52, "Developing Web Services Vusers."

### IDE Integration

To create a script with IDE integration, you write a script in your primary development language and then incorporate it into your environment, such as LoadRunner. You can create a script in your Java or .NET development environment, that emulates your Web Service.

The advantage of this approach is that you can develop a script within your standard work environment. This method usually handles correlation and parameterization automatically. If not, you can implement correlation manually.

Other advantages are that IDE integration also handles advanced and proprietary technologies, which are not handled by WSDL scanning and the recording methods. IDE integration provides an emulation of the real application behavior.

The primary disadvantage of IDE integration is the lack of scalability. If the tests are written in code that is CPU intensive such as Java or .Net, you are limited in the number of Vusers that you can run in a load test. In addition, the IDE integration may require more manpower and development time from your software and QA engineers.

## Performing a Load Test

To successfully test a Web Service, you need to run a load test which checks the service as part of a complete environment. After running the test, you check the results and determine the location of the problems.

The initial testing step is defining the business processes that are typical for your users. You then need to determine how to invoke these processes and generate the scripts (recording, scanning a WSDL, or IDE Integration).

The next step is to plan how many users to emulate and which business processes to execute. This should be defined according to your system requirements. In distributed Web Services configurations, you should also consider which business processes to run within the same load test.

A final, but indispensable step, is setting monitors throughout the system to collect the information during the test run. Before configuring the monitors, you should define the measurements that you want to collect. Mercury provides monitors for a wide range of environments.

## Client Emulation

VuGen supports emulation for several Web Service toolkits, such as .NET version 1.1, Axis version 1.1, Glue version 4.1.2, and MS SOAP version 3.0. Each toolkit interprets the Web Service in a slightly different way. To emulate this unique behavior, VuGen uses several elements as differentiators, representing how the server creates messages such as argument type, format, prefix, and headers.

For information on how to select a toolkit emulation, see "Setting Web Service Run-Time Settings" on page 768.

# 52

# Developing Web Services Vusers

You use VuGen to create a Web Services script by recording a SOAP session or scanning a WSDL document. When you run the script, Vusers emulate real users communicating with the Web Service.

This chapter describes:

- About Web Services
- Getting Started with Web Services in VuGen
- Using the Workflow Wizard for Web Services Scripts
- Creating a New Web Services Script
- Recording a Web Services Script
- Scanning WSDL Documents
- Managing WSDL Documents
- Setting WSDL Validation and Comparison Options
- Editing an XML Tree
- IDE Integration

# About Web Services

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

# Getting Started with Web Services in VuGen

This section provides an overview of the process of developing a Web Services script using VuGen.

**To develop a Web Services script:**

**1 Create an empty Web Services script.**

Use the Workflow Wizard or the standard menus to create a new script, choosing the Web Services Vuser type.

**2 Generate the script.**

Generate a script by running the Web Services wizard to record a Web Services session or scan a WSDL document. For information on selecting the most appropriate method, see "Creating a New Web Services Script" on page 739. For recording, set the recording options (**Tools > Recording Options**). When scanning a WSDL file, run the Validation utility. For more information, see "Managing WSDL Documents" on page 754.

To create a script using IDE integration, see "IDE Integration" on page 765.

**3 Enhance the script.**

Enhance the script by inserting transactions, rendezvous points, and XML functions.

For details, see Chapter 7, "Enhancing Vuser Scripts" and Chapter 77, "Programming with the XML API."

**4 Configure the Run-Time settings.**

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

For information about the run-time settings, see "Setting Web Service Run-Time Settings" on page 768 and Chapter 12, "Configuring Run-Time Settings."

**5 Save and run the script in VuGen.**

Save and run the script in VuGen to verify that it runs correctly. While you record, VuGen creates a series of configuration, data, and source code files. These files contain run-time and setup information. VuGen saves these files along with the script.

For scripts created with WSDL scans, run a WSDL comparison to make sure nothing has changed in the original file. For more information, see "Comparing WSDL Files" on page 769.

For details about running the script as a standalone test, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

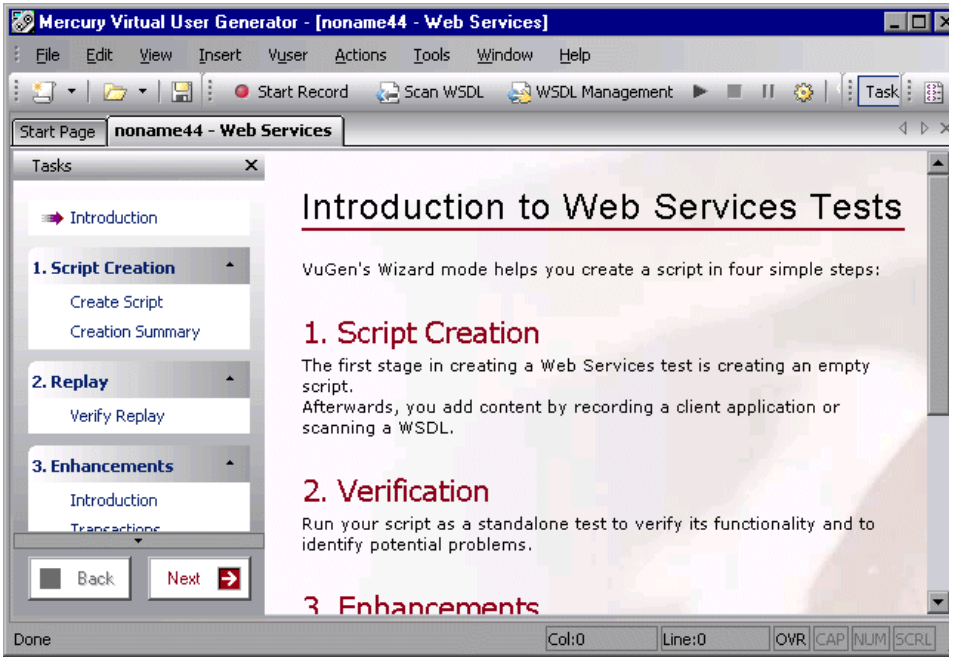
**6 View the test results.**

Open the Test Results utility to view a summary of the replay, for each iteration. For more information, see "Viewing Web Services Reports" on page 785.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# Using the Workflow Wizard for Web Services Scripts

The Workflow screens guide you through the steps of creating a script. By clicking on a link in the Tasks pane, you can read about the steps in creating a script, and view information about your recording and replay. Use the **Back** and **Next** buttons to navigate between screens.




If you do not see the Workflow Wizard, make sure that the Tasks pane is open. (You show and hide the Task pane using the **Tasks** button on the toolbar. Then, click the first link, **Introduction**.

For more information, see Chapter 3, "Using the Workflow Wizard."

The wizard screens that are specific to Web Services are Create Script and Creation Summary.

### Create Script

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

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

### Creation Summary

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

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

- **Protocols** - lists which protocols were used during the script creation
- **Actions** - describes into which sections actions were recorded or scanned.
- **Modify Script** - describes how to proceed in order to modify the script.

## Creating a New Web Services Script

You can create a Web Services Script in one of the following ways:

- Recording a Web Services Script
- Scanning WSDL Documents
- IDE Integration

Recording a client is usually the easiest way to create scripts that emulate real user behavior for a specific business process.

Scanning a WSDL file allows you import a WSDL file to and create a test that communicates with your Web Service.

For IDE integration, you use your regular development environment to write a script. The script, written in a development environment such as VS .NET, emulates a business process in the application's native language.

For more information about choosing the best method for testing, see "Choosing a Web Services Script Type" on page 730.

# Recording a Web Services Script

You can create a Web Services Script by recording a Web Service client.

When you record an application, you can optionally provide a WSDL file describing the Web Service. This allows VuGen to create high-level code, by parsing the recorded SOAP traffic and extracting the calls to the Web Service.

The wizard steps for recording are:

- Specify WSDL File for Recording (only if including a WSDL in the script)
- WSDL Files Validation Summary (only if including a WSDL in the script)
- Specify Application to Record

## Specify WSDL File for Recording

In this screen, you specify the WSDL files. When you record an application, you can record it with or without a WSDL file. Import a WSDL file to create high-level code that parses the recorded SOAP traffic and extracts the calls to the Web Service.

**Do not use WSDL files during recording:** VuGen creates a script with **soap_request** functions.

**Use the specified WSDL files:** Use the Add button to add items to the list. VuGen creates a script with **web_service_call** functions.

For more information on recording, see "Recording a Web Services Script" on page 740.

## Specify Application to Record

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

**Record Web Browser:** Records the traffic of the default Web browser, beginning with the specified URL.

**Record any application:** Records the actions of a specific client application. Specify the full path of the application in the **Program to Record** box. Specify any relevant arguments and working directories.

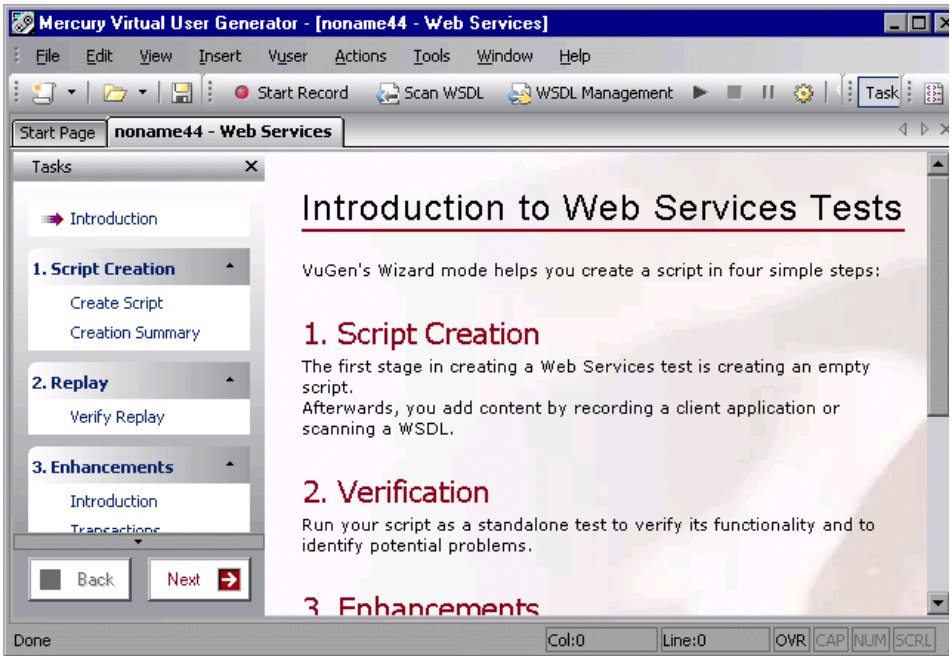
**Record into Action:** The action in which to generate the code. If there are startup procedures that you do not need to repeat, place them in the **vuser_init** section. During recording you can switch to another section, such as **Action**.

**Record Application Startup:** Records the application startup as part of the script. If you want to begin recording at a specific point, not including the startup, clear this option.

For more information on recording, see "Recording a Web Services Script" on page 740.

**To record a Web Services application:**

1 Choose **File > New** to create a new script. Choose the **Web Services** Vuser type from the E-Business Vuser category. VuGen opens the Workflow Wizard.




If you do not see the Workflow Wizard, click the **Introduction** link in the Tasks pane. (Show and Hide the Task pane using the **Tasks** button on the toolbar.)

2 Click the **Create Script** link in the Tasks pane. In the right pane, click on the **Web Services Wizard** button to begin creating your script.

3 Select **Record Client Application** and click **Next**. The wizard moves to the next step: **Specify WSDL files for recording**.

4 To record the application without WSDL files, select **Do not use WSDL files during recording**. Click **Next** and skip to step 6 to specify an application.

5 To record a script with one or more WSDL files, select **Use the specified WSDL files**. VuGen displays a historical list of the WSDL files used. To add a WSDL file to the list:



Click the **Add item** button. VuGen adds a new line to the list.

Type in the path or URL of the WSDL. Alternatively, click the Browse button to the right of the field, and locate the desired file. VuGen adds the new location to the list, with an enabled status.



To permanently remove an entry from the list, select it and click the **Delete**button. To disable a WSDL entry temporarily, clear the check box to the left of the entry.

Click **Next**. VuGen opens the Specifying Application to Record screen.

6 For Web-based applications, select **Record Default Web Browser** and specify a URL to record.

For other Windows applications, select **Record any application**. Enter or browse for the complete path of the client application to record. Optionally, provide program arguments and a working directory.

Choose an action from the **Record into Action** box, and select **Record application startup** to record the invoking of the application.

7 Click **Finish** to close the wizard. The recording toolbar opens. VuGen invokes the client application and begins the recording session.

## Scanning WSDL Documents

You can create a Vuser script by scanning a WSDL document. You specify a path or URL and indicate whether you want to assign values to the methods, or allow VuGen to assign automatic values. You can also mark which methods to include in the script—you do not need to use all of the methods defined in the WSDL.

When scanning a WSDL, VuGen automatically generates **web_service_call** functions for each of the methods. VuGen adds a think time of 3 seconds between each of the method calls.

While scanning a WSDL file, VuGen validates the XML code and syntax of the document. If there is an error or warning, VuGen issues a pop-up message with that information. You can also use the WSDL validation as a stand-alone utility to check the WSDL before creating a script. For more information, see "Managing WSDL Documents" on page 754.

The WSDL Scan wizard lets you control the way VuGen creates a Web Services script. The wizard steps for scanning a WSDL are:

- Specify WSDL for Scanning
- WSDL Files Validation Summary
- Select Methods to Include in Test
- Specify Argument Values
- Final Wizard Screen

### Specify WSDL for Scanning

In this screen, you specify the path or the URL of the WSDL file that you want to scan. VuGen creates a script calling the methods that are defined in the WSDL.

**URL:** A URL of the WSDL file.

**File:** The complete path of the WSDL file on a local or network drive.

**Create Automatic Test:** Creates a simple script that calls all of the WSDL's methods, assigning automatic values to all the elements. If you want to exclude certain methods or manually assign values, proceed through the wizard without creating an Automatic Test.

For more information on scanning, see "Managing WSDL Documents" on page 754.

### WSDL Files Validation Summary

After VuGen successfully scans the WSDL file, the wizard opens the WSDL Files Validation Summary screen. It indicates whether or not the WSDL file has errors.

**Open Validation Report:** Opens the Validation report. This view lets you view the text of the WSDL document and any validation errors or warnings.

---

**Note:** When VuGen scans a WSDL file, it makes a working copy that is saved with the script. By default, all validations and modifications are done on the working copy. If you want to refresh the WSDL file, rescan it using the Scan WSDL wizard.

---

### Select Methods to Include in Test

In this screen, you specify the methods of the WSDL that you want to test. VuGen will create **web_service_call** functions for each one of the methods.

**Service Name:** The name of the Web Service as defined in the WSDL. Select the Web Service whose methods you want to test.

**Description:** A description of the current Web Service.

**Available Methods:** A list of all of the methods in the selected Web Service.

**Selected Methods:** A list of the Web Service methods that will be included in the test. To add methods to this list, double-click on the method or select it in the left pane and click the right-facing arrow.

For more information, see "Managing WSDL Documents" on page 754.

### Specify Argument Values

In this screen, you specify the values of the method's arguments. The right pane changes, depending on the level of the tree node you select.

Note that when you modify a value (input, output, or SOAP header), its icon color changes to blue. In the case of input arguments and headers, this indicates the actual value that VuGen will send to the server.

| If you select... | The right pane shows... |
|---|---|
| A simple input argument | The **Name** of each method and its **Value** |

| | |
|---|---|
| A complex input argument | The following options:<br>• **Include argument in call:** Includes the argument's values in the **web_service_call** functions generated for this argument.<br>• **XML:** Enables the **Import XML** and **Edit XML** buttons. By editing the XML, you can manually insert argument values. See "Editing an XML Tree" on page 764.<br>• **Generate auto-value for this argument:** Inserts automatic values for all arguments of this complex type node.<br>• two additional buttons: **Add** and **Delete**. These buttons allow you to add and delete array arguments by their index. |
| An individual argument | The following information:<br>• **Value:** The value of the argument.<br>• **Generate auto-value for this argument**: Insert automatic values for this node. |
| Output Arguments | The **Name** of the arguments and the **Parameter** that can store the value. |
| SOAP header | An edit box to indicate the value of the SOAP header for the current element. For more information, see "Using Soap Headers" on page 746. |

To modify Soap Headers, see the next section, "Using Soap Headers" on page 746.

## Using Soap Headers

This view is available when you select **SOAP header** in the method's tree view. The right pane lets you indicate whether or not to use SOAP headers. To use them, select **Use SOAP header**. Note that you must individually specify SOAP headers for each element. You can import XML code for the SOAP header, or compose your own using the Edit XML option. For more information, see "Editing an XML Tree" on page 764.

For additional information on scanning a WSDL, see "Scanning WSDL Documents" on page 743.

### Final Wizard Screen

The final wizard screen informs you that you have provided all of the necessary information to generate a script through the scanning of a WSDL document.

Click **Finish** to generate the script.

If you want to set run-time settings before running the script, click the **Run-Time Settings** button. These settings indicate the think time, iterations and run logic during script execution.

## Creating a Script by Importing a WSDL Document

The following procedure explains how to create a script by importing a WSDL file instead of recording an application.

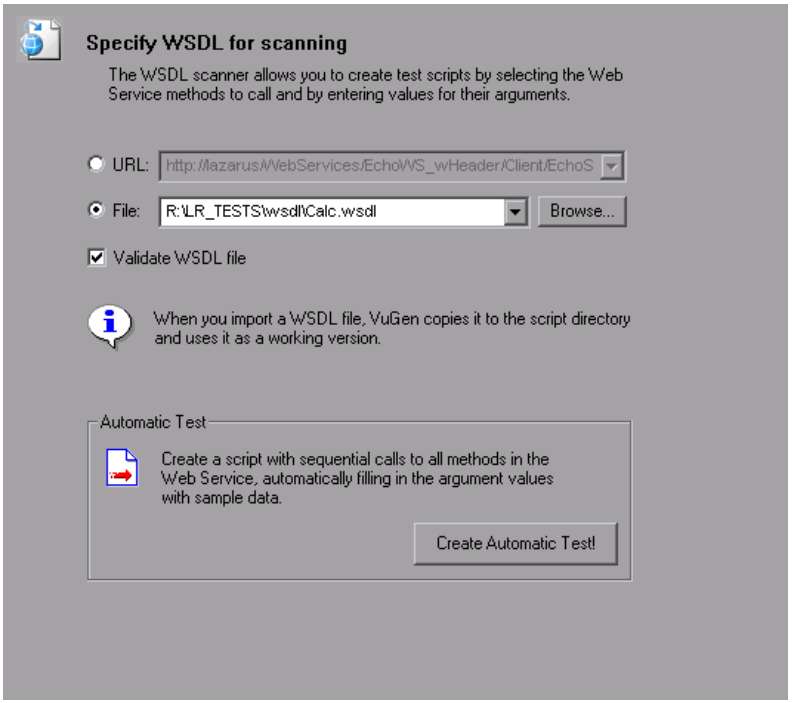
**To create a script by scanning a WSDL document:**

1. Create a new Web Services Vuser Script. Choose the **Web Services** Vuser type from the E-Business Vuser category. VuGen opens the main wizard screen.
2. Select **Scan WSDL file** and click **Next**. The wizard moves to the next step—**Specify WSDL for scanning**.

   If you have already generated a script and you want to scan a new WSDL to an existing script, choose **Vuser** > **Scan WSDL** or click the Scan WSDL button. The WSDL Scan wizard opens to the **Specify WSDL for scanning** step.

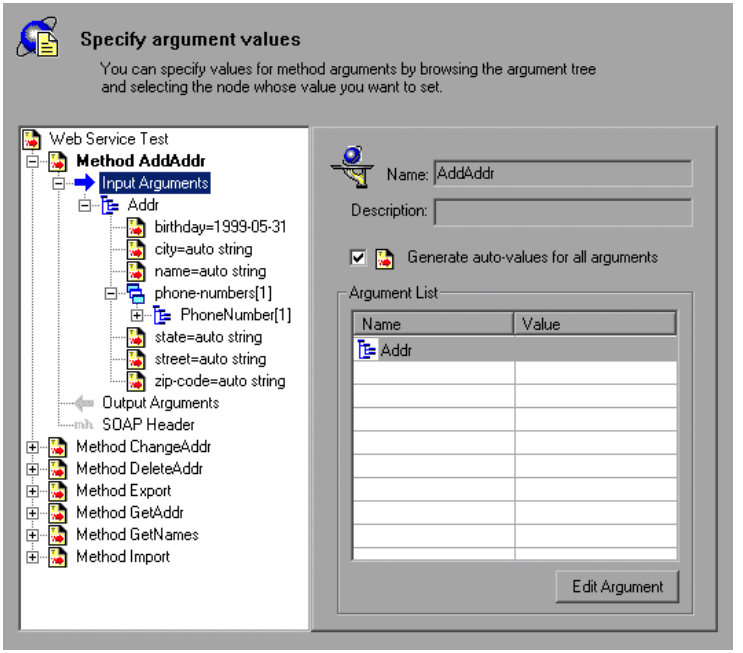


**3** Select **URL** to specify a URL for the WSDL document file. Select **File** to specify or browse for a WSDL on a local or network drive.




**4** To validate the WSDL file during the scanning, select **Validate WSDL file**. If you choose not to validate it now, you can use the WSDL Management window to validate it at a later stage.

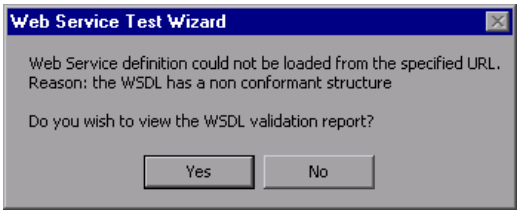
**5** To create a script with input values already assigned, click **Create Automatic Test**. The wizard moves to the next step, assigning automatic values to all of the elements.




If you chose **Create Automatic Test**, proceed to the next step.

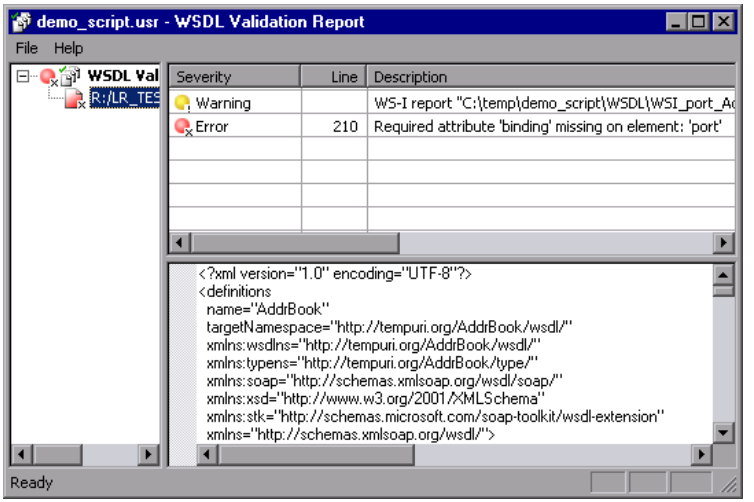
**6** If you want to manually indicate which methods to use, without creating an automatic test, click **Next**. If VuGen succeeds in importing the WSDL file, the WSDL files validation summary step opens.

If VuGen detects a problem with your WSDL, it issues an alert describing the problem.

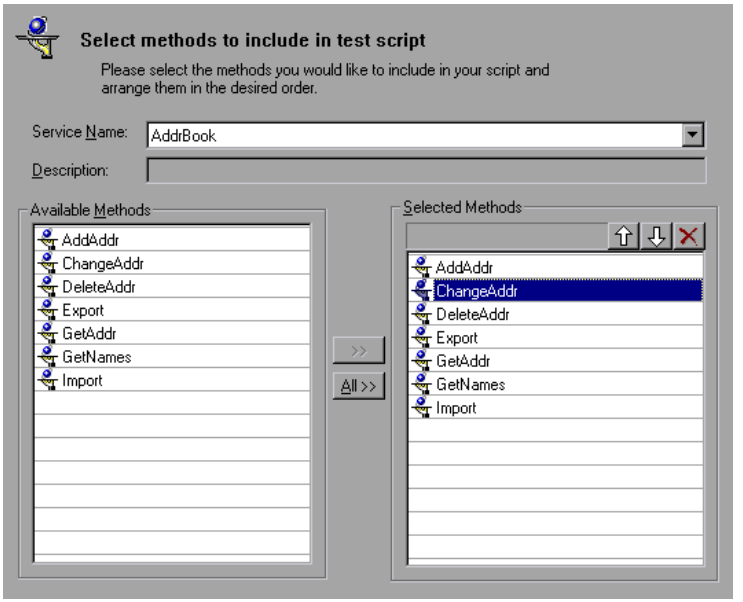


VuGen prompts you to open the Validation report to view the errors or warnings. Click **Yes** to open the Validation report.

**7** If the report is not open, click **Open Validation Report** in the current wizard screen.



**8** Click **Next** in the Scan wizard. It opens the **Select methods to include in test script** step and lists all of the available methods for each of the objects.




In the **Service Name** list, select the desired Web Service.

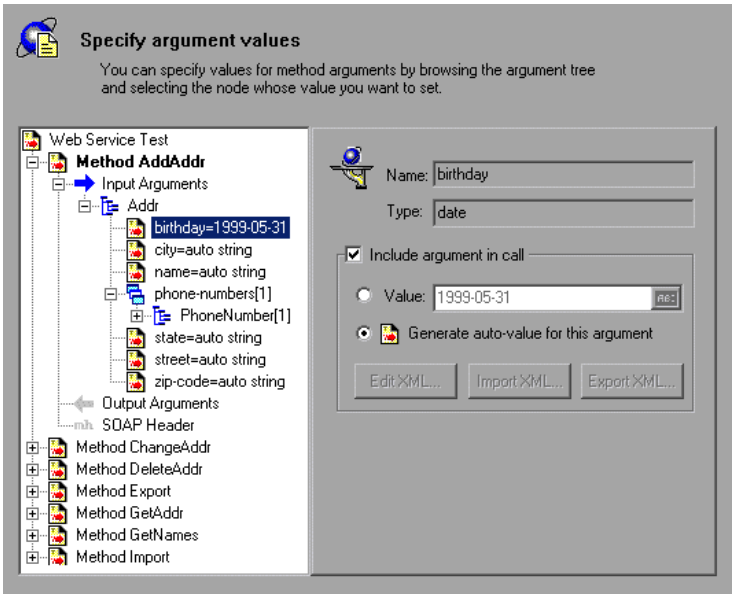
Select one or more methods from the **Available Methods** frame and click the right arrow to move it into the **Selected Methods** section. Use the standard Windows combination keys to select multiple methods.

Click the up or down arrows to rearrange the order of the methods. Click the **Delete** button, **(X)** to delete a method.

Click **Next**. The next step opens—**Specify argument values**.

**9** Expand the methods whose values you want to set. To set the value of an individual argument, select it in the left pane, select the **Value** option in the right pane, and enter a value.

If you want VuGen to automatically generate a value for this argument, select **Generate auto-value for this argument**.

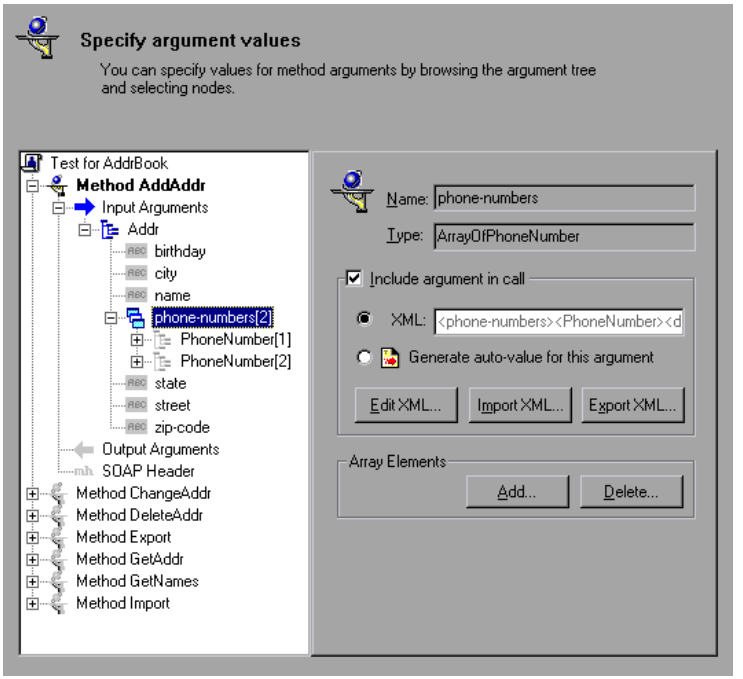



To create a parameter for the argument, click the ABC icon in the right corner of the **Value** box, to open the Select or Create Parameter dialog box.

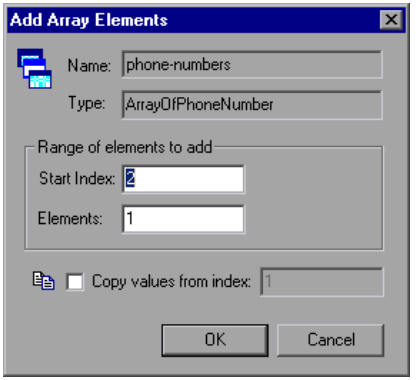
To exclude an argument from the method call, clear the **Include argument in call** option.

To modify complex types and arrays, select the entry and click **Edit XML**. Click **Import XML** to import XML, or **Export XML** to export the selected entry to a separate XML file. For more information, see “Editing an XML Tree” on page 764.

**10** To work with an array, click on it in the left pane.




To add array elements, either simple or complex, click **Add** in the Array Elements section. The Add Array Elements dialog box opens.




Specify a starting index and the number of elements to add.

To assign values of an existing array element to the new elements, select **Copy values from index** and specify the array index of the element whose value you want to use.

Click **OK**. VuGen creates the elements in their complete structure.

To set values for individual array elements, select the arguments and insert the values.

To delete array elements, click **Delete** and specify the starting index and the number of elements to remove.

**11** To view all of the values of the arguments in a method, select the method in the left pane and click **View Arguments**. The right pane displays a list of all of the input arguments and their values. The list shows only the simple argument values—not the complex ones.

**12** Click **Next**. The wizard informs you that you have provided all of the information. To run the script immediately, select **Run script after generation**. To view or modify the run-time settings, click Run-Time settings. Note that it is not mandatory to modify these settings.

**13** Click **Finish**. The wizard places the generated code in VuGen's editor. It creates separate method calls for each one of the selected methods.

**14** Save the script.

## Managing WSDL Documents

VuGen's WSDL Management window lets you add WSDL files to VuGen's working list. When you add a file to the list, VuGen creates a working copy that it saves with the script. VuGen provides several management utilities. You can set options for each of these utilities both before and after creating a script.

The WSDL Management window, assists you in:

- Validating WSDL Files
- Comparing WSDL Files
- Setting WSDL Validation and Comparison Options

## Validating WSDL Files

You can validate the WSDL while creating a script, or validate it independently before creating a script. The Validation report lists all errors, warnings, and notifications relating to the structure or content of the WSDL files. VuGen checks the following items:

- XML form: Is the WSDL code in well-formed XML format?
- WSDL Schema: Does the WSDL conform to the WSDL schema—does it contain the mandatory attributes? Are Namespaces specified when required? Are the Imports and Include files available?

You can instruct VuGen to do validations on several levels. For more information, see "Setting WSDL Validation and Comparison Options" on page 760.

VuGen also lets you test your Web Service for WS-I (Web Services Interoperability) compliance. You can choose a platform and indicate the extent of details for the WS-I compliance test. For more information, see "WS-I Validation Options" on page 762.

**To validate a WSDL file independently (not while creating a script):**

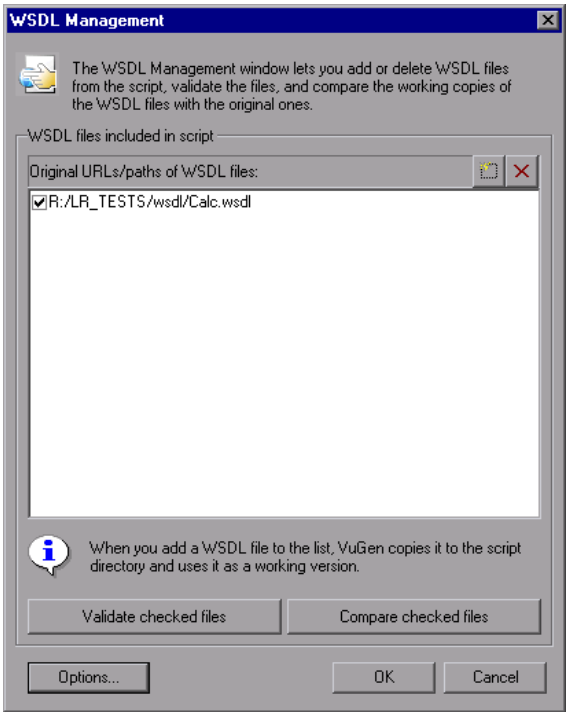
**1 Open a Web Services Vuser script.**

Open a new or existing Web Services type script.

**2 Manage the WSDL files.**




Click the **WSDL Management** button or choose **Vuser > WSDL Management**. The WSDL Management dialog box opens.




**3 Add WSDL files.**



Click the **Add Item** button in the top right corner of the list, to locate and add WSDL files to the list of files to validate.

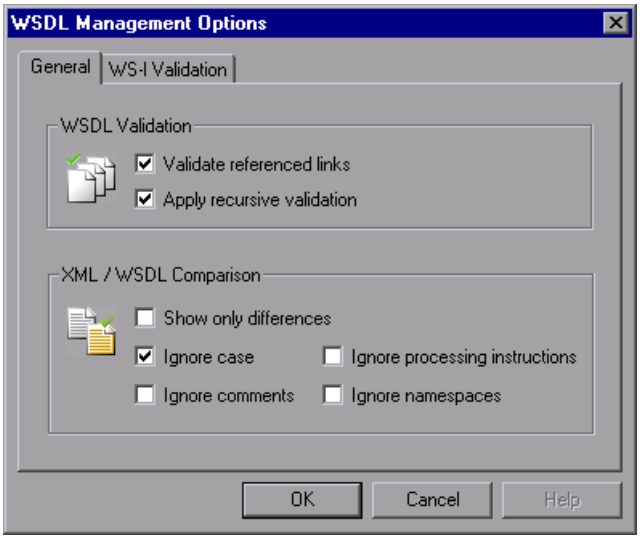
---

**Note:** When you add a WSDL file to the list, VuGen makes a working copy and saves it with the script. By default, all validations are done on the working copy. If you want to refresh the WSDL file, rescan it.

---

**4 Set the validation level.**

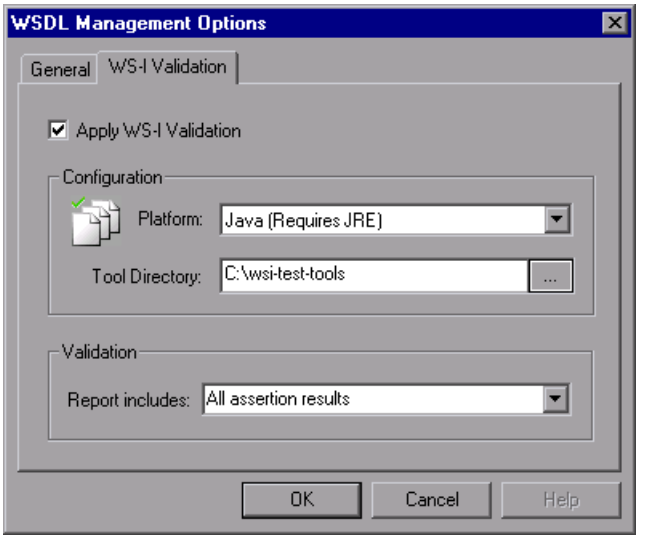
Click the **Options** button. The WSDL Management Options dialog box opens.




Select the desired levels in the WSDL Validation section.

**5 Set the WS-I validation options.**

Click the **WS-I Validation** tab. To enable WS-I validation, select **Apply WS-I Validation**. Choose a platform and enter the directory of the WS-I validation tool. Choose a report type from the **Report includes** list. Click **OK**.

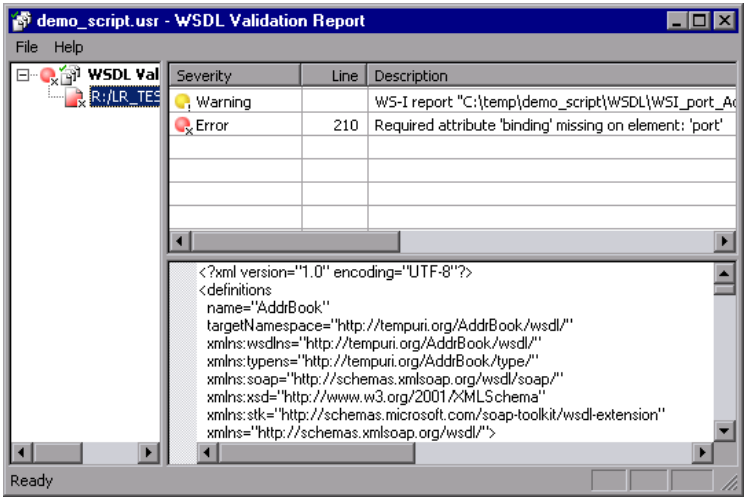



**6 Select the WSDL files to validate.**

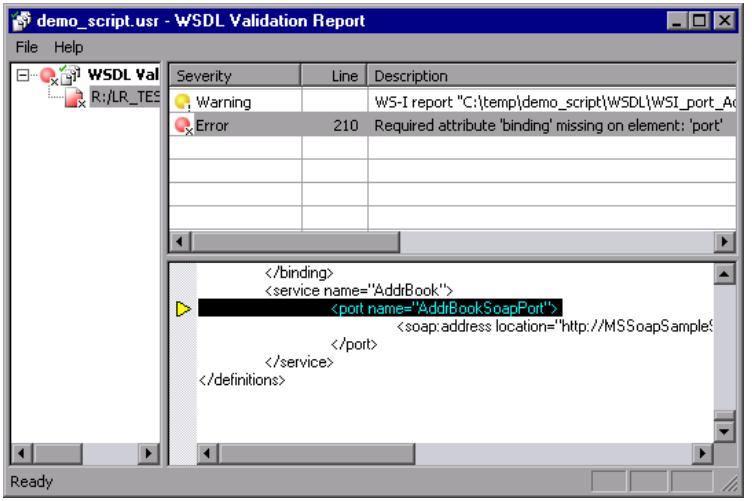
In the WSDL Management window, make sure there are check marks in the boxes adjacent to each of the WSDL files that you want to validate. To add or remove check marks, click in the box.

**7 Validate the files.**

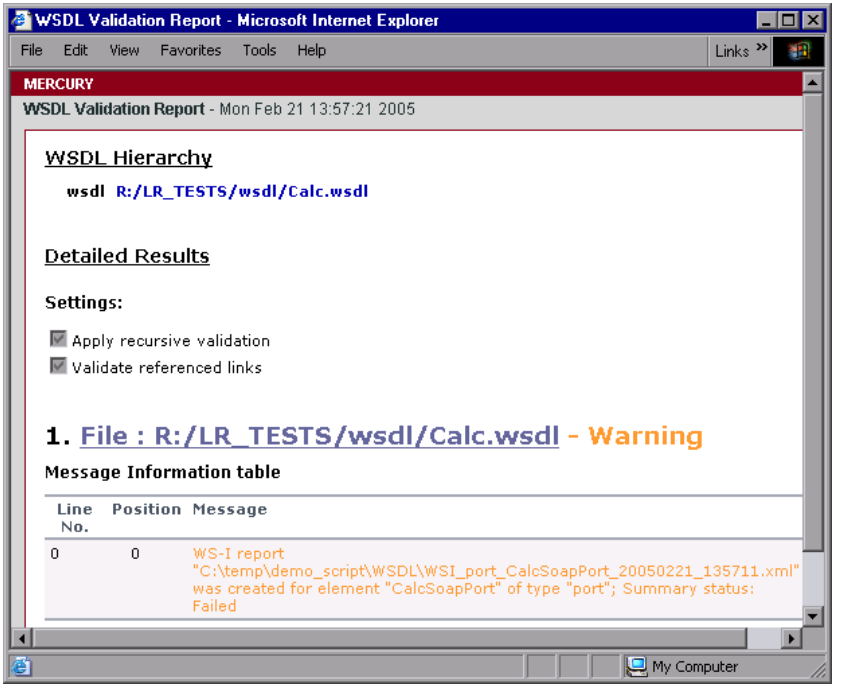
Click **Validate checked files**. The WSDL Validation Report opens.




If there is an error in an WSDL file, VuGen displays its details in the right pane. Click the error (in the top right pane) to locate it in the XML WSDL file (in the bottom right pane).



To open an HTML summary report of the validation, choose **File** > **Export to HTML**. VuGen opens a browser with an HTML report of the Validation results.




8 To save the HTML file, choose **File** > **Save As** from the browser window.

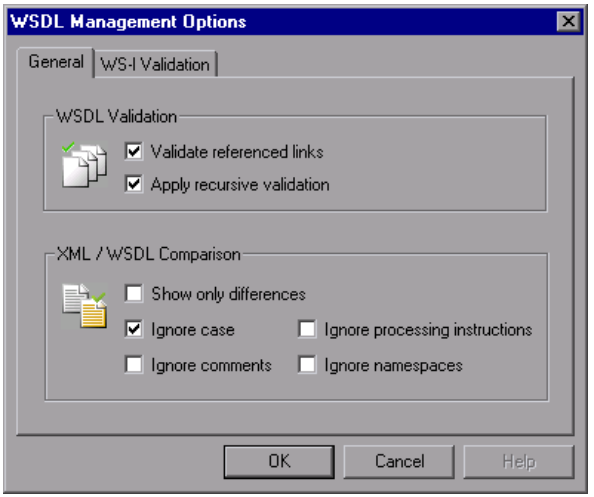
9 To close the WSDL Validation Report window, choose **File** > **Exit**.

## Setting WSDL Validation and Comparison Options

You can set options for WSDL Management in the following areas:

- WSDL Validation Options
- WSDL Comparison Options
- WS-I Validation Options

To view the options, click **Options** in the WSDL Management window. (**Vuser** > **WSDL Management**).




### WSDL Validation Options

You can validate the WSDL while creating a script, or independently using the WSDL Management window. This step checks the XML code in the WSDL document and makes sure that it is valid according to the WSDL standards.

VuGen offers three levels of validation for WSDL files:

- **Simple** - Validates only the XML code, but none of the imported or included files. For this level, do not select any of the check boxes.
- **Validate Referenced Links** - Validates the XML code and verifies that the imported or included files exist and that they are valid.
- **Apply Recursive Validation** - Validates all XML code and imported or included. It also validates the XML code at the referenced imported or included files, as well as all items referenced by them.

### WSDL Comparison Options

VuGen offers several comparison options when comparing the local and global copies of the WSDL documents:

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

## WS-I Validation Options

WS-I (Web Services Interoperability) is an organization that created standards for Web Services, to promote compatibility across platforms, operating systems, and programming languages.

### Obtaining the Testing Tool

VuGen lets you check WS-I compliance with tools provided by WS-I. You can download the tools from WS-I's Website at http://www.ws-i.org/.

Download the tool that matches your platform: C# with .NET or Sun Java. Note that there are also several versions of the testing tools, such as version 1.0 or 1.1. After you download the zip file, extract the files to a local drive, maintaining the original directory structure, **wsi-test-tools**.

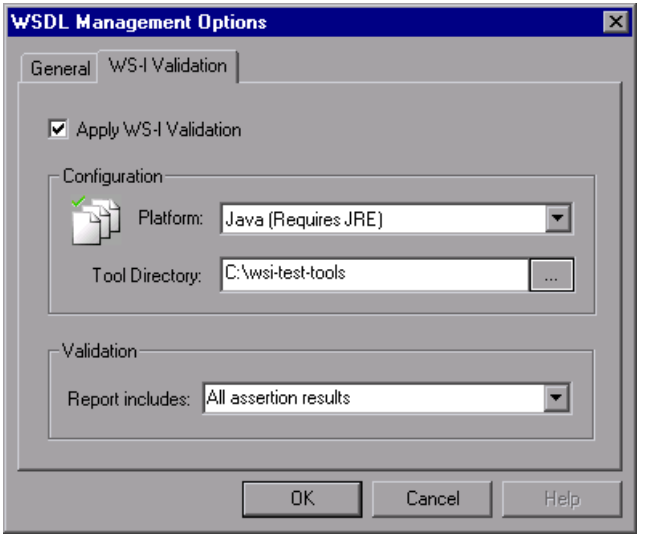
If you define an environment variable called WSI_HOME with a value of the path of **wsi-test-tools**, VuGen will recognize the path and automatically activate WS-I validation for WSDL. If you do not define an environment variable, you can manually enable WS-I validation and browse to the location of the WS-I testing tool.

### WS-I Validation Reports and Logs

After you validate your Web Service for WSDL, you can view both WSDL and WSI validation reports

The WSI report is an XML file which contains information about the WSDL's compliance with WS-I.

To enable WS-I validation when generating a validation report, click **Options** in the WSDL Management window and click the **WS-I Validation** tab. Select the **Apply WS-I validation** option.




**Platform:** The platform of your Web Service: Microsoft .NET, which requires the .NET Framework, or Java (Sun Java).

**Tool Directory:** The path of the WS-I validation tool, wsi-testing-tool.

**Validation Options:** Specify which messages to include in the report:

- **All assertion results**: Show all results.
- **Assertions with the result "Failed"**: Show only the assertions with a "Failed" result status.
- **Assertions with a Result different than "Passed"**: Show the assertions that did not have a "Passed" result status.

To open the WSI validation report, use the right-click menu from within the WSDL validation report.

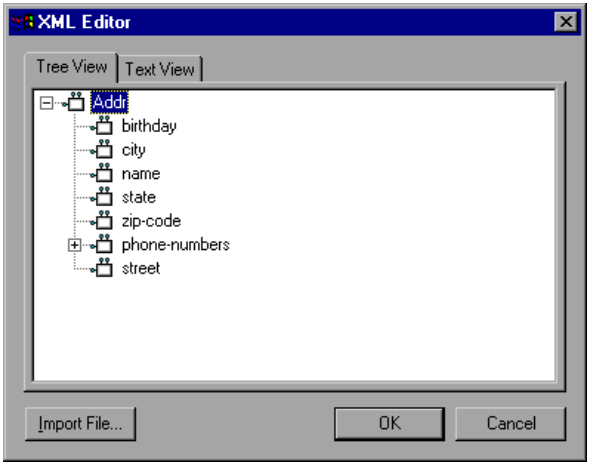
# Editing an XML Tree

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

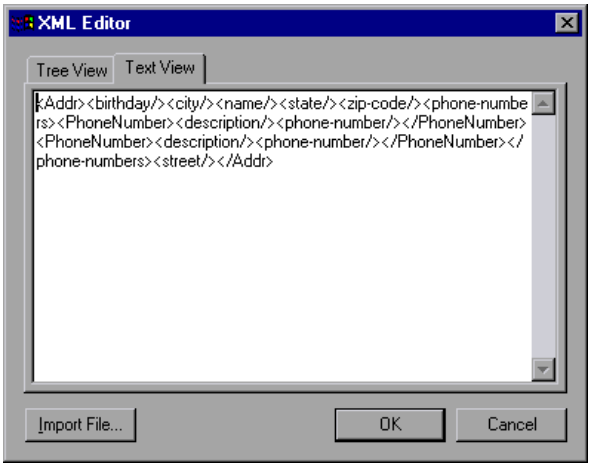
Entering the values for the XML elements is a tedious and error-prone task. VuGen provides you with an interface that simplifies the task of entering, saving, and restoring the information. Once you enter the data manually, you can save it to an XML file using the **export** option. For subsequent tests,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

**To work with XML strings:**

1. Select a complex type or array in the left pane.
2. To edit the XML code for that entry, click **Edit XML**. The XML Editor window opens.



**3** To edit the code in text mode, click the **Text View** tab. Edit the XML code manually.




**4** To import a previously saved XML file, click **Import File** and specify the file's location. Edit the file in the XML Editor window.

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

## IDE Integration

The IDE integration method allows you to use components from your client application to create the scripts. For example, if your Web Service uses WS security or another propriety implementation for asynchronous calls, recording this traffic would not generate a meaningful script. In this case, you create a script with the actual client application code. You run the client directly in a load test environment without recording it.

This script creation process requires coordination between the QA and software developers. The tests that the developers create for testing the APIs internally, can usually be used for load testing too.

# 53

# Running Web Services Vusers

After you create a Web Services script, you run it to make sure it is functional. After you run the script, you can view test results to see whether the script succeeded in communicating with the Web Service.

This chapter describes:

- About Running Web Services Vusers
- Setting Web Service Run-Time Settings
- Comparing WSDL Files
- Comparing XML Files
- Setting Scanned WSDL Properties
- Viewing Web Services Script Snapshots
- Using Web Services Functions
- Viewing Web Services Reports

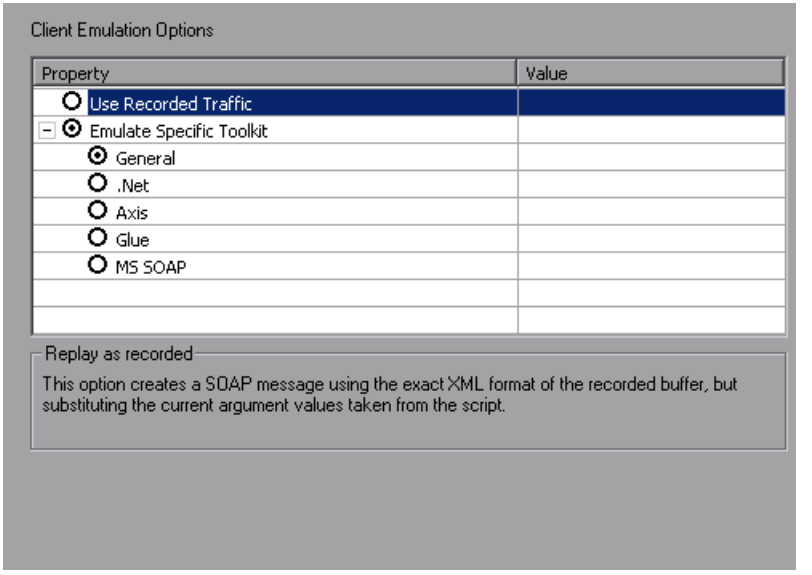
## About Running Web Services Vusers

When you run the script, Vusers emulate real users communicating with the Web Service. Before running the script, there are several things you can do to make your script more effective and correct:

- **Run-time settings:** You can set run-time settings that help you emulate real users more accurately.
- **WSDL Comparison:** It is recommended that you check for updates to your WSDL file before you replay the script. This is especially important for a script that you created on an earlier date.

# Setting Web Service Run-Time Settings

You can set the run-time settings for Web Services in order to emulate a client application or a specific toolkit.




For scripts that you created by recording a client application (excluding those you created by scanning a WSDL file), you can emulate the client using the styles and attributes in your recorded script. To use this type of emulation, enable the **Use Recorded Traffic** option.

For scripts created by scanning a WSDL file, only the second option is available—**Emulate specific toolkit**. You can select the type of emulation required. In addition to the specific toolkits that can be emulated, there is also a general type of emulation. For more information about the emulation, see “Client Emulation” on page 733.

**To set the Web Services run-time settings:**

1 Open the Run-Time settings dialog box (**Vuser** > **Run-Time Settings** or F4) and select the **Web Services:Client Emulation** node.

2 Select the desired emulation or the generic emulation (**General**).

# Comparing WSDL Files

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

It is possible, however, that by the time you run the script, the original WSDL file will have changed. This will lead to inaccurate test results. Therefore, before replaying a Web Services script that was created at an earlier date, you should run a comparison test on the WSDL file.

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

If the differences are significant, you can update the WSDL from the original copy using the Refresh option.

VuGen lists the differences between the files in a Comparison report. The Comparison Report window has two columns— **Working Copy** and **Original File.** The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

Note that VuGen also has a general utility that allows you to compare any two XML files. For more information, see "Comparing XML Files" on page 773.

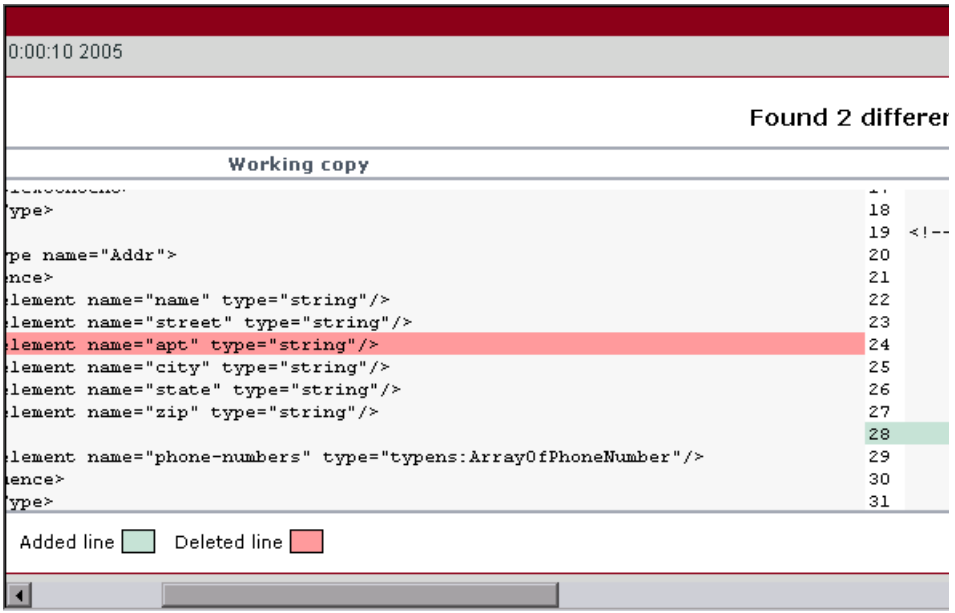
The Comparison report uses the following legend to mark the differences between the two files:

**Yellow**—changes to an existing element (shown in both versions)

**Green**—a new element added (shown in the original file copy)

**Pink**—a deleted element (shown in the working copy)

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




### WSDL Comparison Options

VuGen offers the following comparison options when comparing the local and global copies of the WSDL documents:

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

**To compare WSDL files:**

**1 Open a Web Services Vuser script.**

Open an existing script.

**2 Open the WSDL Management window.**

Click the **WSDL Management** button or choose **Vuser** > **WSDL Management**. The WSDL Management dialog box opens.

**3 Add WSDL files.**



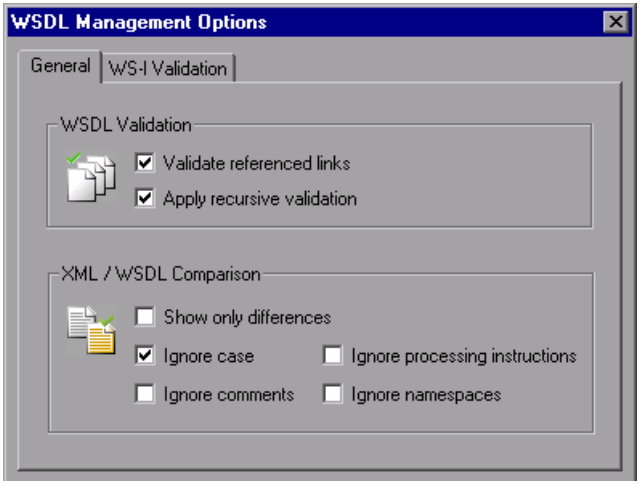
Click the **Add** button to locate and add WSDL files to the list of files to compare.

**4 Select the desired WSDL files to compare.**

Make sure that there is a check mark in the box adjacent to each of the WSDL files that you want to compare.

**5 Set the Comparison Options.**

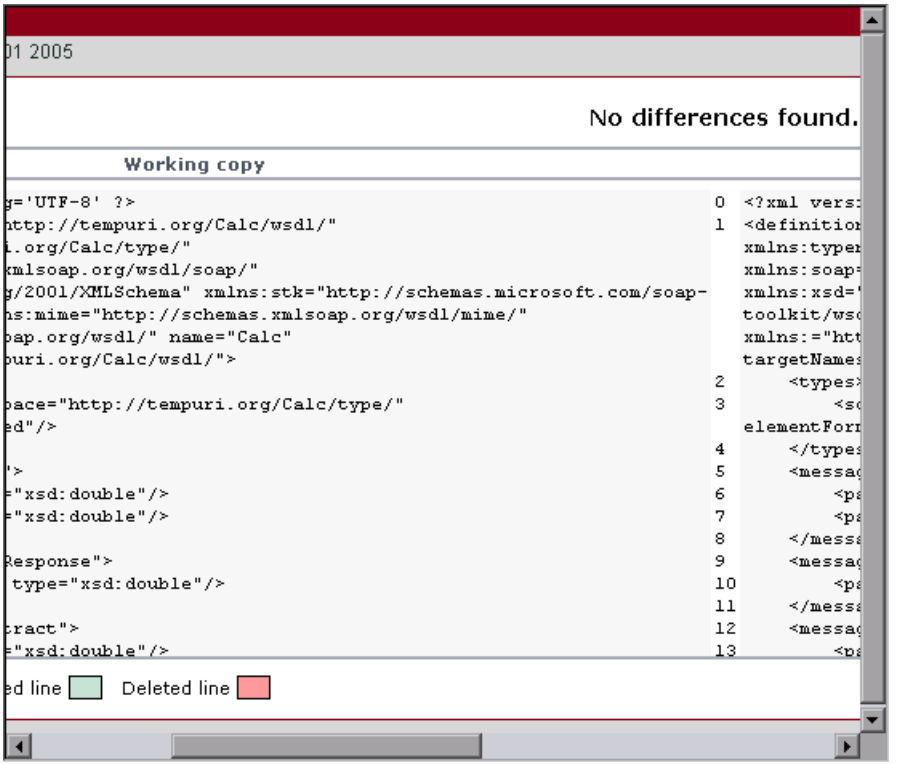
Click the **Options** button. The WSDL Operations Options dialog box opens.




Select the desired options in the General tab, and click **OK**.

**6 Compare the files.**

Click **Compare checked files with originals**. The WSDL Comparison Report opens.




Scroll down through the file to locate the differences.

**7 Refresh the working copy.**

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

**8** To close the WSDL Comparison Report window, choose **File** > **Exit**.

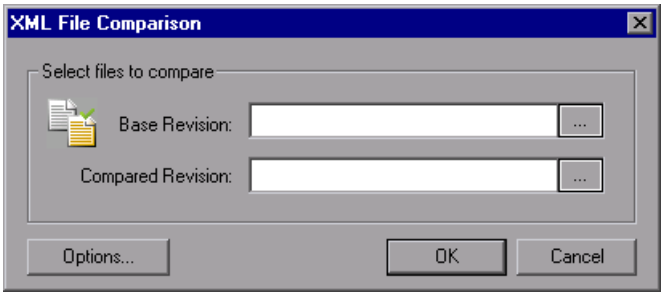
# Comparing XML Files

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

You can specify what differences to ignore, such as case or comments. For additional information about the comparison options, see "WSDL Comparison Options" on page 770.

**To compare two XML files:**

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




2 Click the Browse button to the right of the **Base Revision** box to locate the original XML file.

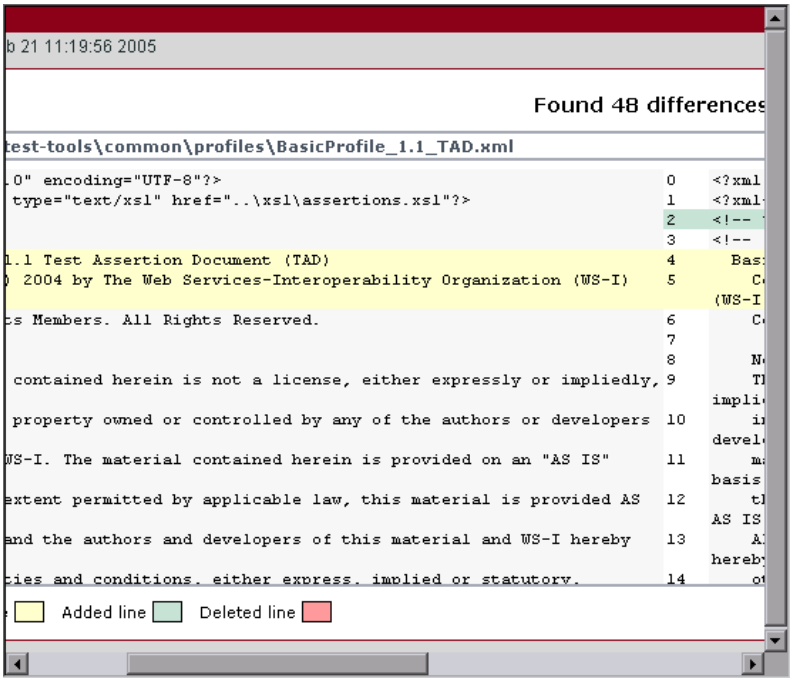
3 Click the Browse button to the right of the **Compared Revision** box to locate the newer XML file.

4 Click **OK**. VuGen opens the XML Comparison Report window.

For information about the Comparison report, see below.

## XML Comparison Reports

VuGen lists the differences between the files in a Comparison report. The Comparison Report window has two columns—the base revision and the compared revision. The header of each column lists the full path of the XML files.




The Comparison report uses the following legend:

**Yellow**—changes to an existing element (shown in both versions)

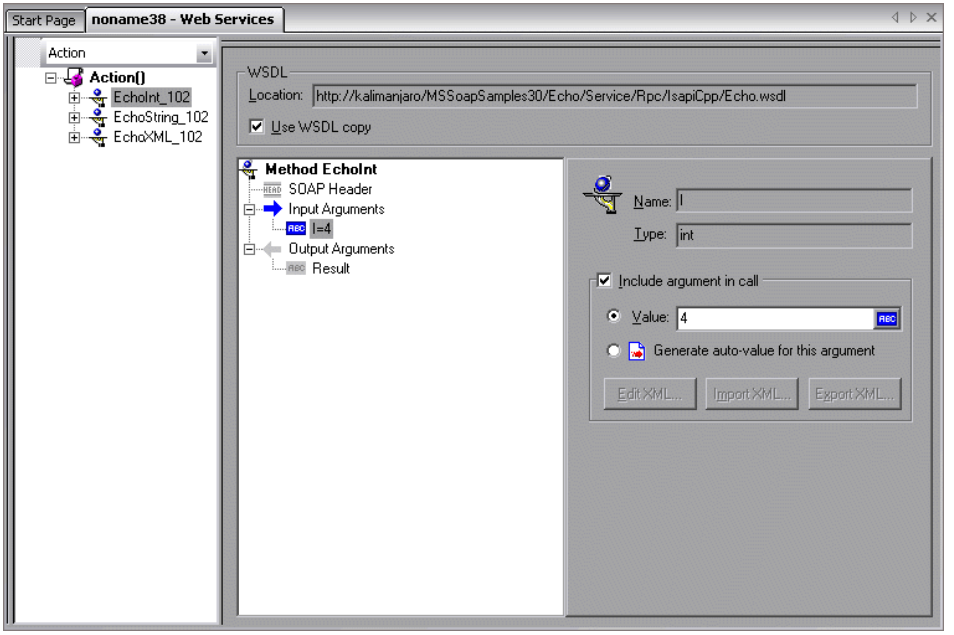
**Green**—a new element added (shown in the global version)

**Pink**—a deleted element (shown in the local version)

## Setting Scanned WSDL Properties

After you scan a WSDL file—without recording, VuGen's Tree view shows the properties of each element in the right pane.

To view the scanned WSDL in Tree view, choose **View > Tree view**.




Modify each argument value as necessary. For more information, see "Specify Argument Values" on page 745.

## Viewing Web Services Script Snapshots

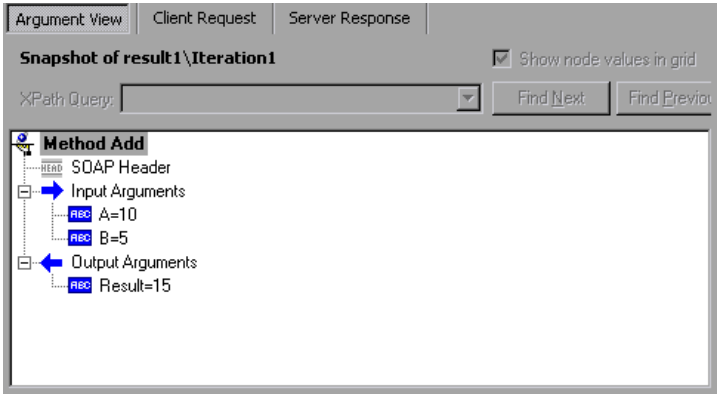
You can use VuGen's snapshot viewer to examine the SOAP requests and responses that occurred during record and replay. Note that you must replay the session at least once in order to view a replay snapshot.

Note that when you scan a script instead of recording it, there is no snapshot. Instead, the right pane shows the method properties. For more information, see "Setting Scanned WSDL Properties".

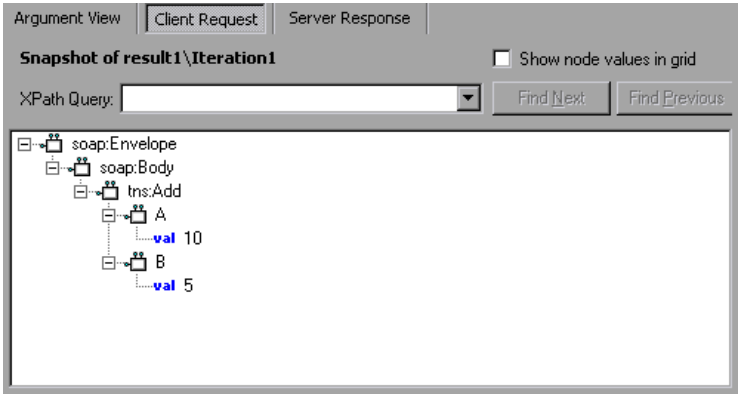
There are several ways to view the snapshot:

- Argument View
- Client Request
- Server Response

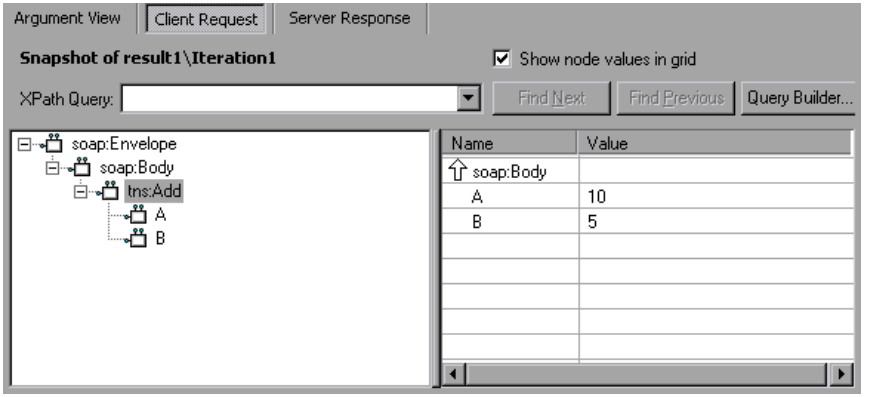
The **Argument View** displays the arguments and their values. This view shows you the values that were sent to the server during the Web Services session.



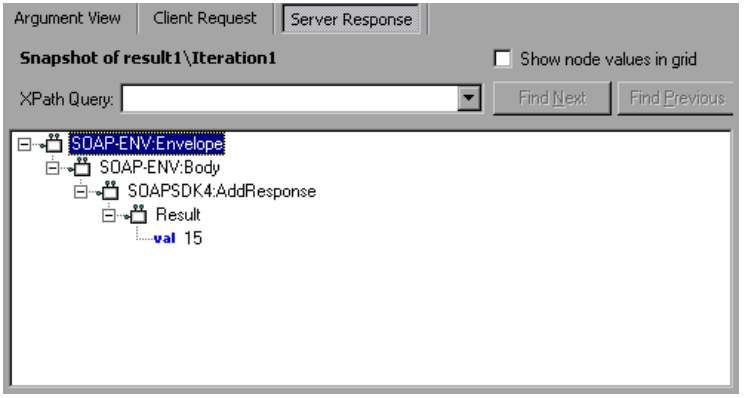

The **Client Request** tab displays the arguments and the values that you assigned to them in an expandable tree hierarchy.



The **Show node values in grid** option also allows you to view the elements and their values in a grid. When working in the Server Response view, you can select a value in the grid and parameterize it using the right-click menu.




The **Server Response** tab displays all the result elements in an expandable tree hierarchy.




Using the **Client Request** or **Server Response** tabs, you can view the XML elements and their properties (using the right-click menu) or click the **Query Builder** button to perform a query on the XML tree.

The following sections describe:

- Querying an XML Tree
- Working With XML Sent By the web_service_call Function
- Parameterizing XML Elements
- Inserting Verification Functions

### Querying an XML Tree

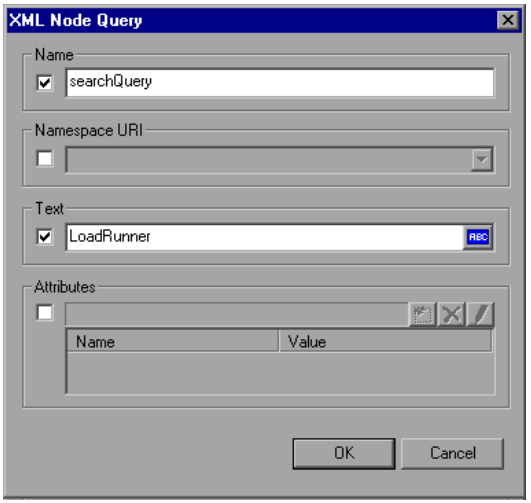
VuGen displays the XML code in an expandable tree. In the case of a SOAP envelope packet, you can see the various elements of the packet and their values. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

**To perform a query:**

1 Click **Query Builder** in the Snapshot window. The XML Node Query dialog box opens.




2 Enable one or more items for searching.



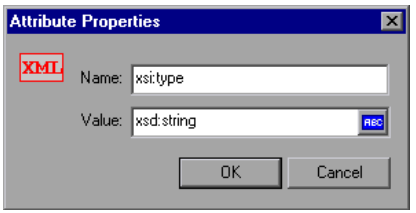
3 Enable the **Name** section to search for the name of a node or element.

4 Enable the **Text** section to search for the value of the element indicated in the Name box.

5 Enable the **Namespace URI** section to search for a Namespace.

6 Enable the **Attributes** section to search for an attribute.

7 Enter the search text in the appropriate boxes. To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.




8 Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the XPath query box.



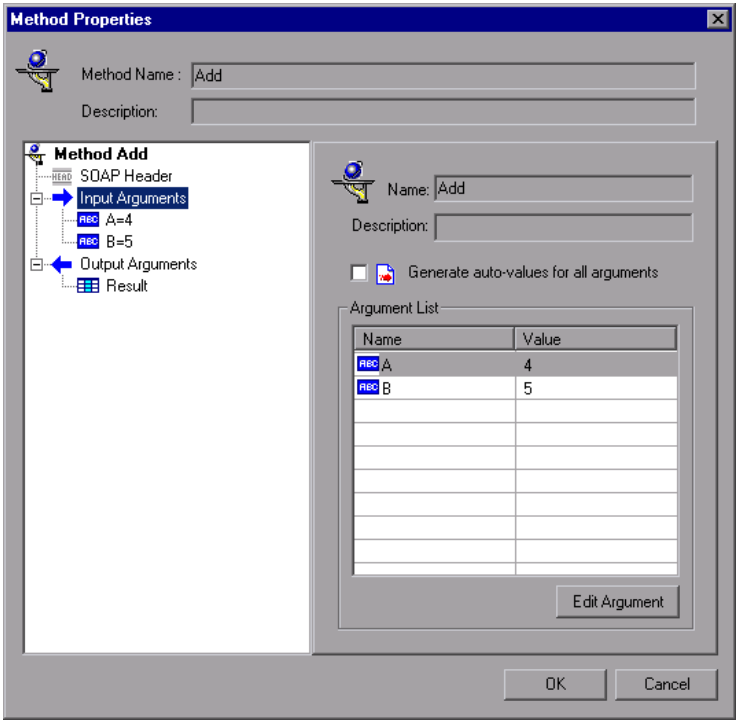

9 Click **Find Next** to begin the search.

## Working With XML Sent By the web_service_call Function

For the **web_service_call** function, you can view the general properties of a Web Service method, or modify its input and output arguments.

**To view or modify properties:**

1. In the tree view, select a step.
2. Double-click on the step or click the **Properties** button at the top of the right pane to display the Properties dialog box.

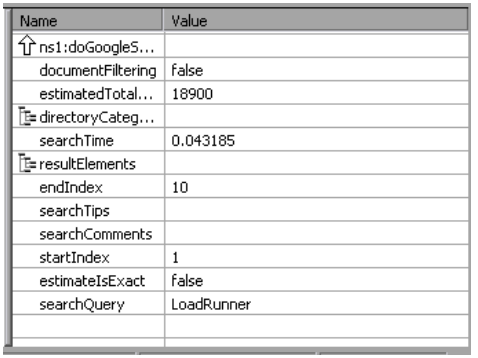


## Parameterizing XML Elements

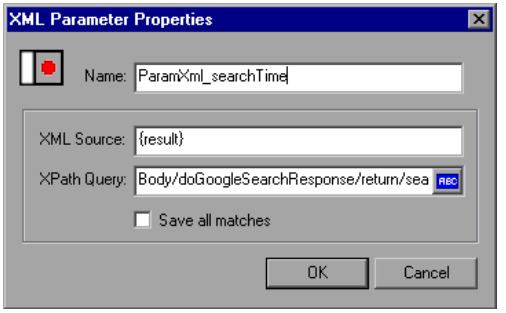
The Snapshot viewer and grid let you parameterize name and text value within the XML document.

**To replace a value with a parameter:**

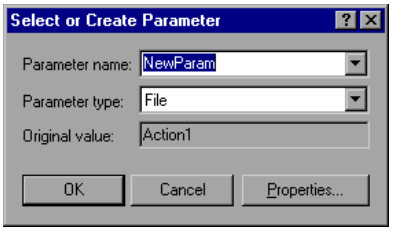
**1** Select the tree node of the argument you want to parameterize, or select the value in the grid.




**2** Select **Save value in parameter** from the right-click menu. The XML Parameter Properties dialog box displays the properties of the selected XML element.



**3** To parameterize the element, click the ABC icon in the **XPath Query** box. The Select or Create Parameter dialog box opens.




**4** Specify a parameter name and type. For more information, refer to the For details, see Chapter 8, "Working with VuGen Parameters."

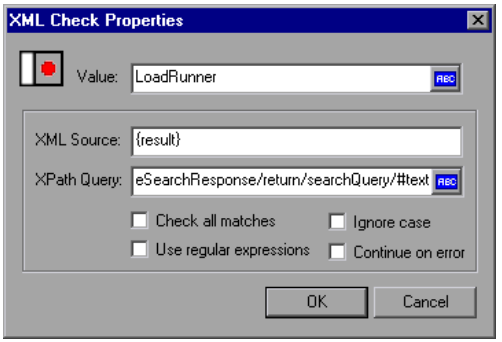
## Inserting Verification Functions

When running a script, you may want to verify that a certain text string is displayed. You perform text verification by selecting the element you want to check, and inserting an XML check step. VuGen places an **lr_xml_find** function as the cursor.

**To insert an XML check:**

**1** In the grid or tree node, select the element you want to check.

**2** Choose **Insert XML Check** from the right-click menu. The XML Check Properties dialog box opens.




Enter the value of the element in the Value box.

**3** Select the desired options:

**Use Regular Expressions** (corresponds to the **UseRegExp** option): Allows the search value to be a regular expression.

**Continue on Error** (corresponds to the **NotFound** option): Continue running the script even if the search value is not found. If cleared, the script fails if the value is not found.

For more information, refer to **lr_xml_find** in the *Online Function Reference* (**Help** > **Function Reference**).

## Using Web Services Functions

When you record a Web Service session or scan a WSDL document, VuGen creates functions that call the Web Service's methods:

- **web_service_call** - When scanning a WSDL document, or recording a Web Services session which includes a WSDL file.
- **soap_request** - When recording a Web Services session without specifying a WSDL.

The following examples illustrate the difference in the code between the two functions:

**web_service_call**

```
web_service_call( "StepName=Add_101",
        "SOAPMethod=Calc.CalcSoapPort.Add",
        "ResponseParam=response",
        "WSDL=R:/LR_TESTS/wsdl/Calc.wsdl",
        "UseWSDLCopy=1",
        "Snapshot=t1108909353.inf",
        BEGIN_ARGUMENTS,
        "A=5",
        "B=6",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
```

The above code describes the WSDL's **Add** method from the **calc.wsdl** file. The result parameters are listed between the BEGIN_RESULT and END_RESULT markers. In this method of code generation, the SOAP Header is not used.

If you enabled the **Include argument in call** option in the Scan wizard, VuGen includes the input arguments between the BEGIN_ARGUMENTS and END_ARGUMENTS markers.

**soap_request**

```
web_add_header("SOAPAction", "http://tem
                        puri.org/Calc/action/Calc.Add");

soap_request( "URL=http://war/MSSoapSamples30/Calc/Ser
                    vice/Rpc/IsapiCpp/Calc.WSDL",
      "SOAPEnvelope=<?xml version=\"1.0\" encoding=\"UTF-8\" standal
      one=\"no\"?><SOAP-ENV:Envelope xmlns:SOAP-ENV=\"http://sche
      mas.xmlsoap.org/soap/envelope/\"><SOAP-ENV:Body><AddNs:Add
      xmlns:AddNs=\"http://tempuri.org/Calc/mes
      sage/\"><A>4</A><B>5</B></AddNs:Add>"
          "</SOAP-ENV:Body></SOAP-ENV:Envelope>",
          "Snapshot=t1.inf",
          "ResponseParam=result1",
          LAST);
```

The above code describes the WSDL's **Add** method from the **calc.WSDL** document as a SOAP request. All information about the method and its input arguments are placed within a SOAP envelope.

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

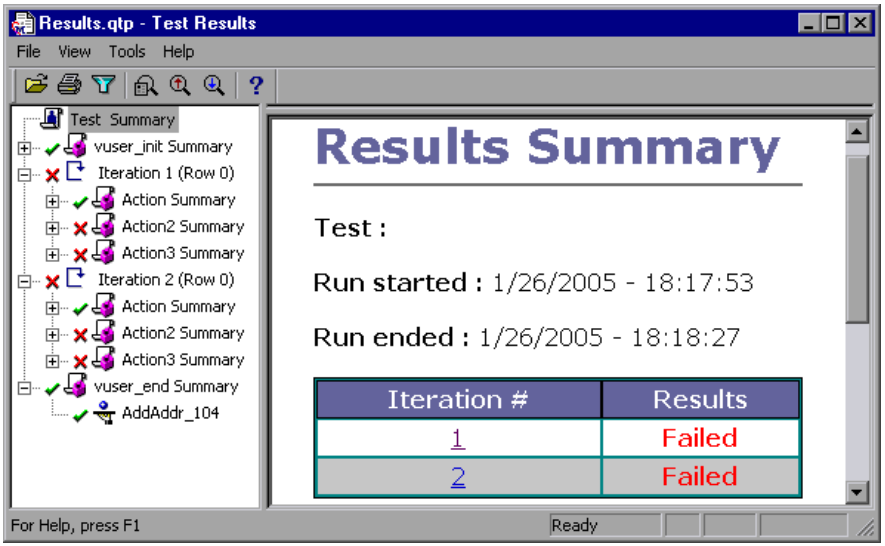
## Viewing Web Services Reports

After you run a Web Services script, you can view a summary of the test results using VuGen's Test Results utility. The Test Results utility shows both Web Vuser steps and Web Services Vuser steps.

This section describes the Summary report's Web Services information. For general information about the Test Results utility and the available views, see Chapter 49, "Using Reports to Debug Vuser Scripts."

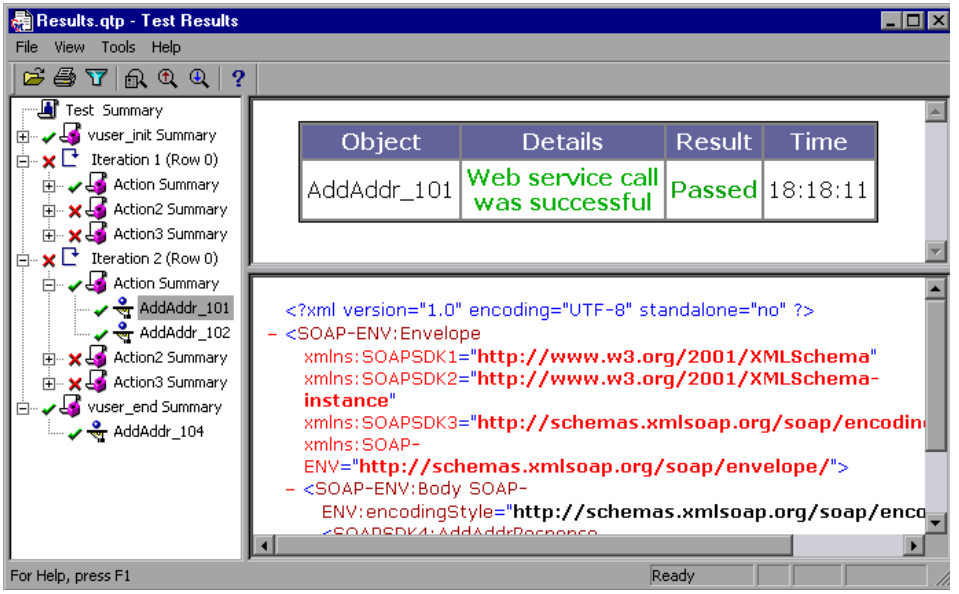
To open the Summary report, choose **View** > **Test Results**.

The test results are divided into iterations, actions, and steps.

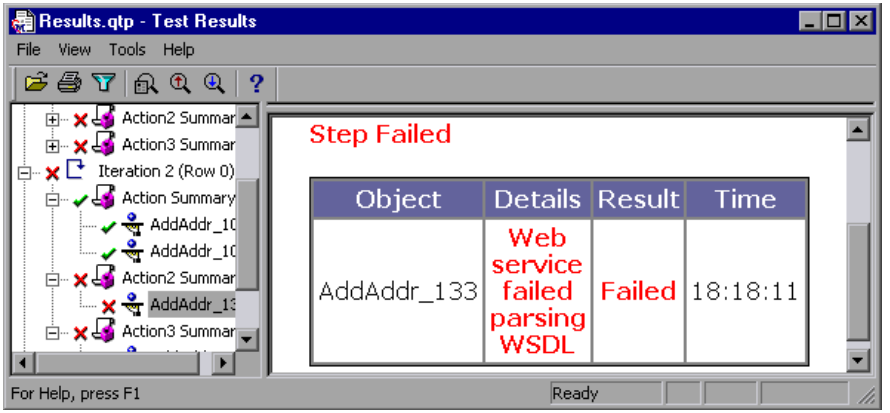



The Results report marks a successful step with a green check mark and a failed step with a red **X**. An iteration is only marked as successful if all of its steps and actions have succeeded.

For Web Service calls, the bottom pane of the Results window displays the contents of the SOAP response.




If VuGen cannot parse or interpret the code, the report contains a message stating **Web service failed parsing WSDL**.




For more information about working with the Test Results, see Chapter 49, “Using Reports to Debug Vuser Scripts.”

# 54

# Recording Web/WinSock 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:

- About Recording Web/WinSock Vuser Scripts
- 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, 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, refer to 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 46, "Setting Correlation Rules for Web Vuser Scripts." For details on correlating WinSock functions, see Chapter 11, "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. Perform typical operations.

For details, see "Setting Browser and Proxy Recording Options" on page 792.

**2 Enhance the script.**

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 7, "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 8, "Working with VuGen Parameters."

4 **Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts", and Chapter 11, "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 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

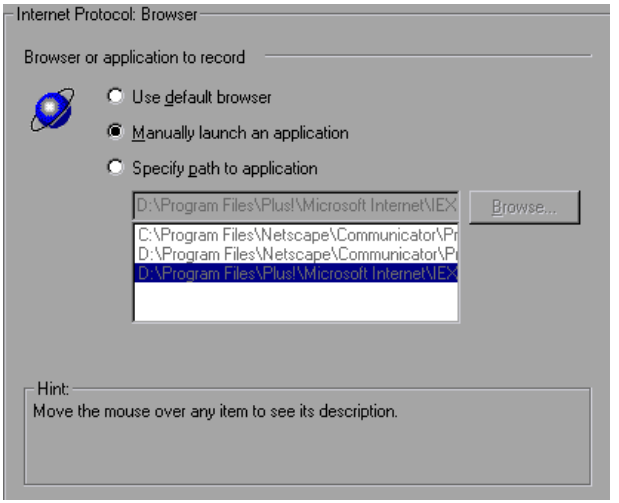
## Setting Browser and Proxy Recording Options

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 40, "Setting Recording Options for Internet Protocols." For information on WinSock recording options, see Chapter 26, "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 796). If you enable Web trapping, the application in the **Program to Record** field is always launched.

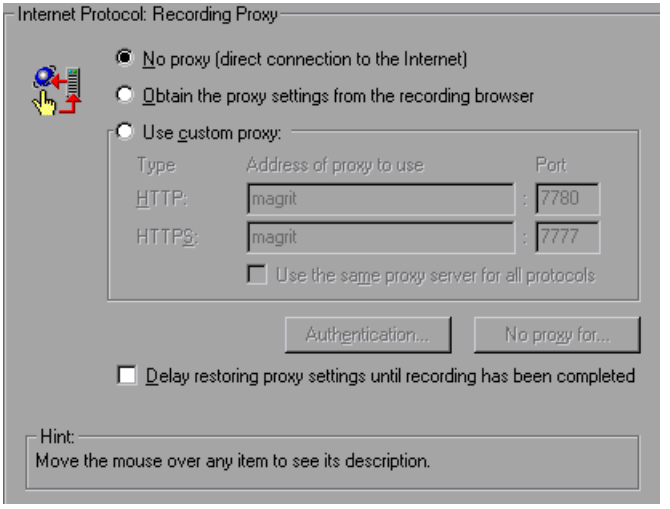
- **Use default browser**, to instruct VuGen to use the default Web browser on the recording computer. The application in the **Program to Record** field of the Start Recording dialog box is ignored. You must, however, enter a value into this field, even though it is not used. Use this option to record 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. VuGen launches the browser when it begins recording and prompts you to modify the proxy settings. 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 an 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 **No proxy** 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 41, "Setting Recording Options for Web Vusers." For information on WinSock recording options, see Chapter 26, "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 793.

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.

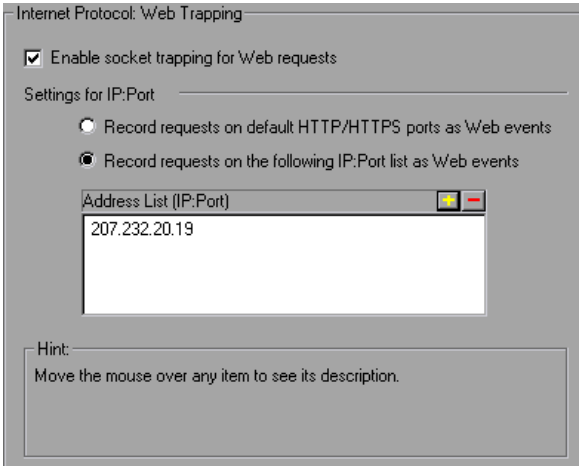
To determine whether or not to enable Web trapping, first perform a recording session. View the data file, **data.ws**. If you see HTTP or HTTPS data that was recorded as WinSock buffer data, this may indicate that the request was made over a different port. In this instance, you should enable Web trapping 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 793.

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 "+" sign to add a new IP:port entry to the list. Click the "-" sign 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:**

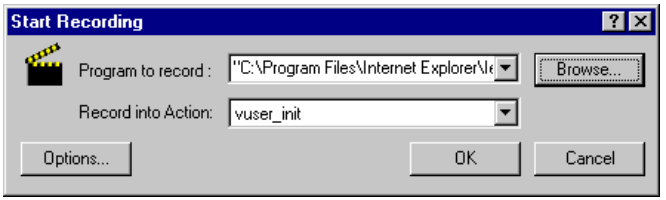
1 Open the recording browser, and set the home page to the URL you want to record.

2 Open the Mercury Virtual User Generator, VuGen, from the **Start** menu.



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**. See "Setting the Browser Recording Options" on page 793 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 44, "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 7, "Enhancing Vuser Scripts."

After modifying a script, you can revert back to the originally recorded version of the script, using the Regenerate Script utility. This utility only regenerates the WinSock statements; it does not affect the Web statements. For more information, see Chapter 4, "Recording with VuGen."

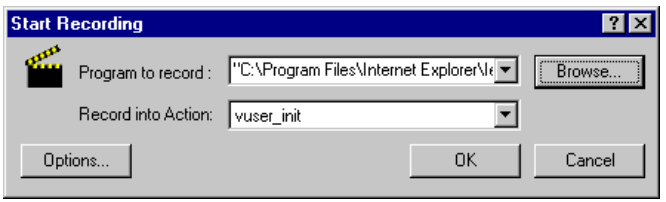
## Recording Palm Applications

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**.

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.

# Part IX

## Enterprise Java Bean Protocols

# 55

# Performing EJB Testing

The Enterprise Java Beans (EJB) testing tool generates scripts for testing or tuning EJB objects.

This chapter describes:

- About EJB Testing
- Working with the EJB Detector
- Creating an EJB Testing Vuser
- Setting EJB Recording Options
- Understanding EJB Vuser Scripts
- Running EJB Vuser Scripts

***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 test or tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. VuGen's EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to test or tune, and VuGen generates a script that emulates each of the EJB's methods.

It creates transactions for each method so that you can measure its performance and locate problems. In addition, each method is wrapped in a **try and catch** block for exception handling.

Note that in order to create EJB testing scripts, the EJB Detector must be installed and active on the application server host. The Detector is described in the following sections.

VuGen also has a built-in utility for inserting methods into your script. Using this utility, you display all of the available packages, select the desired methods, and insert them into your script. For more information, see "Running EJB Vuser Scripts" on page 822.

# Working with the EJB Detector

The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

## Installing the EJB Detector

The EJB Detector can be installed and invoked on the application server's machine or alternatively, on the client machine. To run the EJB Detector on the client machine you must have a mounted drive to the application server machine.

**To install the EJB detector agent:**

1. Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
2. Unzip the **<LR_root>\ejbcomponent\ejbdetector.jar** file into the EJB Detector directory.

## Running the EJB Detector

The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

The EJB Detector can run from the command-line, or from a batch file.

**To run the EJB Detector from the command line:**

1. Before running the EJB Detector from the command line, add the DETECTOR_HOME\classes and the DETECTOR_HOME\classes\xerces.jar to the CLASSPATH environment variable.
2. If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the CLASSPATH:

   For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar

   For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar
3. If your EJBs use additional classes directory or .jar files, add them to the CLASSPATH.
4. To run the EJB Detector from the command-line, use the following string:

   java EJBDetector [search root dir] [listen port]

**search root dir**

One or more directories or files in which to search for EJBs (separated by semicolons). Follow these guidelines:
**BEA WebLogic Servers 4.x and 5.x:** Specify the application server root directory.
**BEA WebLogic Servers 6.x**: Specify full path of the domain folder.
**WebSphere Servers 3.x:** Specify the full path of the deployed EJBs folder.
**WebSphere Servers 4.0:** Specify the application server root directory.
**Oracle OC4J:** Specify the application server root directory.
**Sun J2EE Server:** Specify the full path to the deployable **.ear** file or directory containing a number of **.ear** files.

If unspecified, the classpath will be searched.

**listen port**

The listening port of the EJB Detector.The default port is 2001. If you change this port number, you must also specify it in the **Host name** box of the **Generate EJB Test** dialog box.

For example, if your host is metal, if you are using the default port, you can specify **metal**. If you are using a different port, for example, port 2002, enter **metal:2002**.

**To run the EJB Detector from a batch file:**

You can launch the EJB detector using a batch file, **EJB_Detector.cmd**. This file resides in the root directory of the EJB Detector installation, after you unzip **ejbdetector.jar**.

**1** Open env.cmd in the EJB Detector root directory, and modify the following variables according to your environment:

| | |
|---|---|
| **JAVA_HOME** | the root directory of JDK installation |
| **DETECTOR_INS_DIR** | the root directory of the Detector installation |
| **APP_SERVER_DRIVE** | the drive hosting the application server installation |
| **APP_SERVER_ROOT** | Follow these guidelines:<br>**BEA WebLogic Servers 4.x and 5.x:** Specify the application server root directory.<br>**BEA WebLogic Servers 6.x**: Specify full path of the domain folder.<br>**WebSphere Servers 3.x:** Specify the full path of the deployed EJBs folder.<br>**WebSphere Servers 4.0:** Specify the application server root directory.<br>**Oracle OC4J:** Specify the application server root directory.<br>**Sun J2EE Server:** Specify the full path to the deployable **.ear** file or directory containing a number of **.ear** files. |
| **EJB_DIR_LIST (optional)** | list of directories/files, separated by ';' and containing deployable .ear/.jar files, and any additional classes directory or .jar files or used by your EJBs under test. |

**2** Save **env.cmd**.

**3** If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested, as well as the following vendor EJB classes, to the CLASSPATH in the env file:

For WebLogic 4.x: <WebLogic directory>\lib\weblogicaux.jar

For WebSphere 3.x: <WebSphere directory>\lib\ujc.jar

4 Run the EJB_Detector.cmd or EJB_Detector.sh (Unix platforms) batch file to collect information about the deployable applications containing EJBs, for example:

```
C:\>EJB_Detector [listen_port]
```

where listen_port is an optional argument specifying a port number on which the EJB Detector will listen for incoming requests (default is 2001).

### EJB Detector Output and Log Files

You can examine the output of the EJB Detector to see if it has detected all the active EJBs. The output log shows the paths being checked for EJBs. At the end of the scan, it displays a list of the EJBs that were found, their names and locations.

For Example:

```
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_beanManaged.jar…
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_statefulSession.jar…
Checking EJB Entry: f:/weblogic/myser-
ver/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.

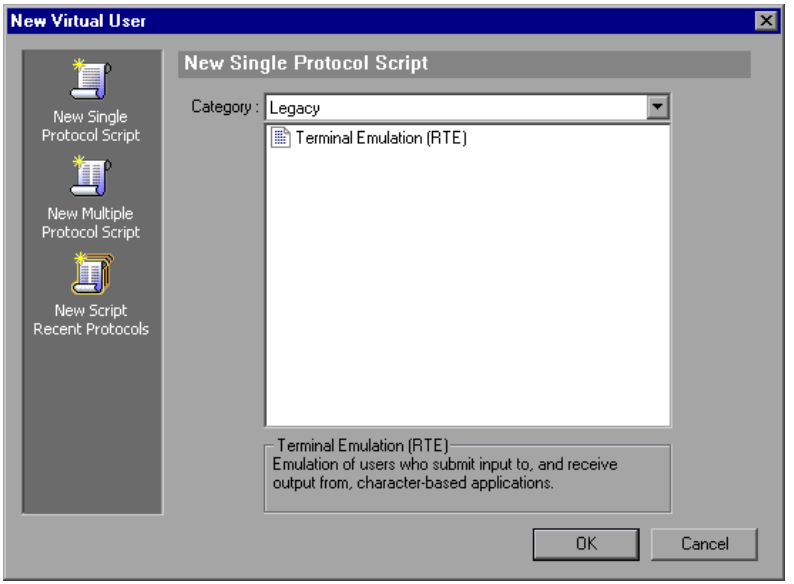
After the EJBs are detected, the HTTP Server is initialized and waits for requests from the VuGen EJB-Testing Vuser. If there are problems in this communication process, enable the property webserver.enableLog in the **webserver.properties** file located in the DETECTOR_HOME\classes directory.

This enables printouts of additional debug information, and other potentially important error messages in the **webserver.log** file.

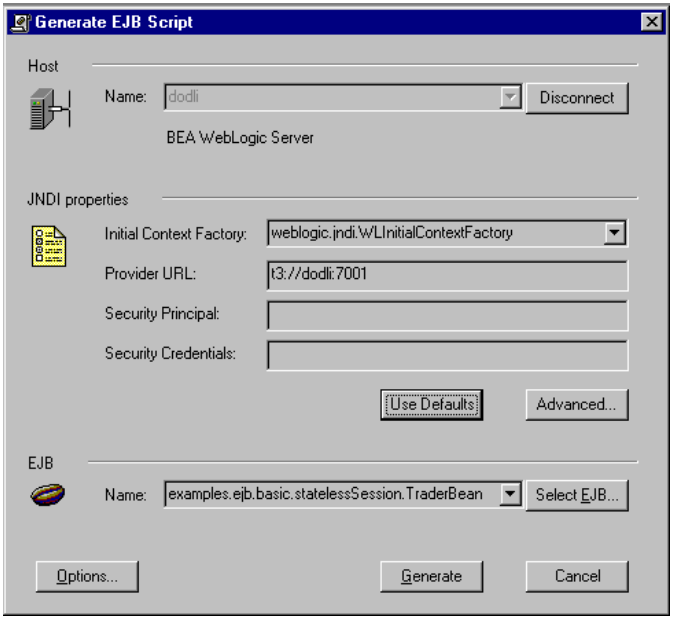
## Creating an EJB Testing Vuser

**To create an EJB Vuser script:**

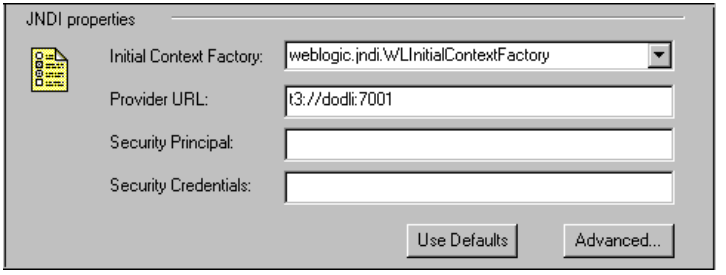
**1** Choose **File** > **New** or click the **New** button. The New Virtual User dialog box opens.



**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 VuGen's EJB Detector is installed. Note that the Detector must be running in order to connect. Click **Connect**. The **JNDI properties** section is enabled.



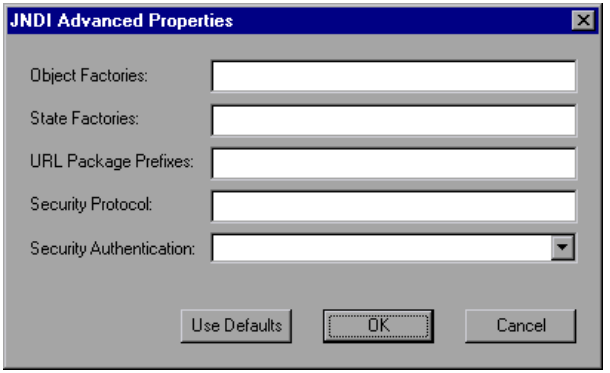
**4** The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory and** the **Provider URL**.

If your application server requires authentication, enter the user name in the **Security Principal** box and a password in the **Security Credentials** box.

Here are the default values of the two JNDI mandatory properties:

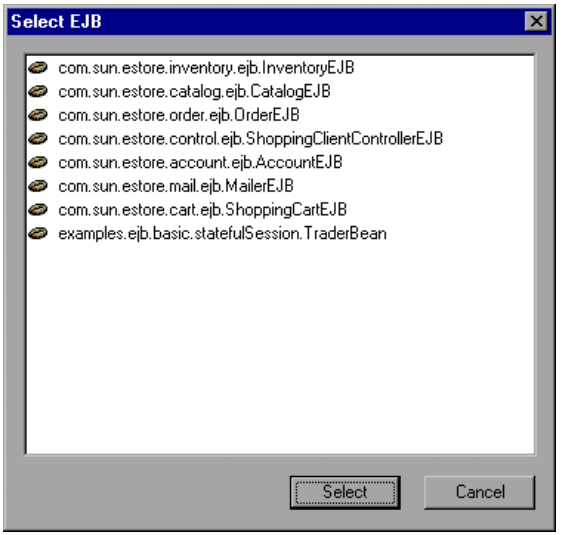
| Type | Initial Context Factory | Provider URL |
|---|---|---|
| WebLogic | weblogic.jndi.WLInitialContextFactory | t3://<appserver_host>:7001 |
| WebSphere 3.x | com.ibm.ejs.ns.jndi.CNInitialContextFactory | iiop://<appserver_host>:900 |
| WebSphere 4.x | com.ibm.websphere.naming.WsnInitialContextFactory | iiop://<appserver_host>:900 |
| Sun J2EE | com.sun.enterprise.naming.SerialInitContextFactory | N/A |
| Oracle | com.evermind.server.AppplicationClientInitialContextFactory | ormi://<appserver_host>/<application_name> (the app. name of the EJB in <oc4j>/config/server.xml) |

**5** To set advanced properties for the JNDI, click **Advanced** to open the JNDI Advanced Properties dialog box.



Specify the desired properties: **Object Factory**, **State Factory**, **URL Package Prefixes**, **Security Protocol**, and **Security Authentication**. Click **OK**.

**6** In the **EJB** section of the dialog box, click **Select** to 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 to test and click **Select**.

**8** In the Generate EJB Script dialog box, click **Generate**. VuGen creates a script with Java Vuser functions. The script contains code that connects to the application server and executes the EJB's methods.

**9** Save the script.

Note that you cannot generate test code for an additional EJB, within an existing script. To create a test for another EJB, open a new script and repeat steps 2-9.

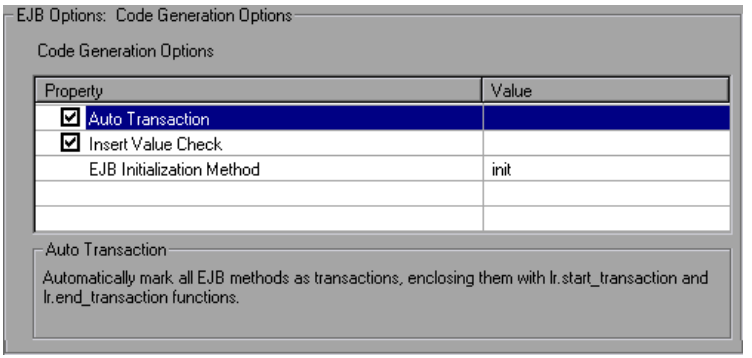
# Setting EJB Recording Options

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 18, "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.Con-
text.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
			p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001");
			javax.naming.InitialContext _context = new javax.nam-
ing.InitialContext(p);

			// lookup Home Interface in the JNDI context and narrow it
			Object homeobj = _context.lookup("carmel.CarmelHome");
			_carmelhome = (CarmelHome)javax.rmi.PortableRemo-
teObject.narrow(homeobj, CarmelHome.class);

		} catch (javax.naming.NamingException e) {
			e.printStackTrace();
		}
```

---

**Note:** If the script is generated with an EJB Detector running on the client rather than an application server, you must manually modify the URL of the provider. For example, in the following line, the provider specifies dod as the EJB detector host name:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001")
```

Replace the recorded host name with the application server name, for example:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://bealogic:7001")
```

You can specify the provider URL before recording, so you don't have to modify it manually, in the **JDNI Properties** section of the Generate EJB Script dialog.

---

## Creating an Instance

Before executing the EJB methods, the script creates a Bean instance for the EJB. The creation of the instance is marked as a transaction to allow it to be analyzed after the script is executed. In addition, the process of creating an instance is wrapped in a **try and catch** block providing exception handling.

**For Session Beans** - use the EJB home 'create' method to get a new EJB instance.

In the following example, the script creates an instance for the Carmel EJB.

```
// create Bean instance
Carmel _carmel = null;
try {
        lr.start_transaction("create");
        _carmel = _carmelhome.create();
        lr.end_transaction("create", lr.AUTO);
} catch (Throwable t) {
        lr.end_transaction("create", lr.FAIL);
        t.printStackTrace();
}
```

**For Entity Beans** - use the findByPrimaryKey method to locate the EJB instance in an existing database, and if not found, then use the create method, to create it there.

In the following example, the script attempts to locate an instance for the account EJB, and if it fails then creates it.

```
// find Bean instance
try {
	com.ibm.ejs.doc.account.AccountKey _accountkey = new
com.ibm.ejs.doc.account.AccountKey();
	_accountkey.accountId = (long)0;

	lr.start_transaction("findByPrimaryKey");
	_account = _accounthome.findByPrimaryKey(_accountkey);
	lr.end_transaction("findByPrimaryKey", lr.AUTO);
} catch (Throwable thr) {

	lr.end_transaction("findByPrimaryKey", lr.FAIL);
	lr.message("Couldn't locate the EJB object using a primary key.
Attempting to manually create the object... ["+thr+"]");

	// create Bean instance
	try {
		lr.start_transaction("create");
		_account = _accounthome.cre-
ate((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 820).
3. Insert additional methods, and modify their default values. For more information, refer to the section on inserting Java functions in Chapter 17, "Recording Java Language Vuser Scripts."
4. Set the general run-time settings for the script. For more information, see Chapter 12, "Configuring Run-Time Settings."
5. Set the Java VM run-time settings: Specify all additional classpaths and additional VM parameters. Make sure to include your application server EJB classes. The actual classes of the EJB under test are saved in the Vuser directory and retrieved automatically during replay. For information about specifying additional classpaths and setting the Java VM run-time settings, see Chapter 20, "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\iiimp.jar

➤ Copy the following files from the <WS>\jdk\jre\lib\ext folder to the <JDK>\jre\lib\ext directory: iioprt.jar, rmiorb.jar.

➤ Copy the 'ujc.jar' from the <WS>\lib folder, to <JDK>\jre\lib\ext.

➤ Copy the file <WS>\jdk\jre\bin\ioser12.dll to the <JDK>\jre\bin folder.

where <*WS*> is the home folder of the WebSphere installation and <*JDK*> is the home folder of the JDK installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

**7** For **WebSphere 4.0** users:

Make sure that you have valid Java environment on your machine of IBM JDK1.3. Open the Run-Time Settings dialog box and select the **Java VM** node. Add the following entries to the **Additional Classpath** section:

```
<WS>/lib/webshpere.jar;
<WS>/lib/j2ee.jar;
```

Where <WS> is the home directory of the WebSphere installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

---

**Note:** If your application server is installed on a UNIX machine or if you are using WebSphere 3.0.x, you must install IBM JDK 1.2.x on the client machine to obtain the required files.

---

**8** For **Oracle OC4J** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher (JDK1.3 preferable). Open the Run-Time Settings dialog box and select the **Java VM** node. Add the following entries to the **Additional Classpath** section:

```
<OC4J>/orion.jar;<OC4J>/ejb.jar;<OC4J>/jndi.jar; ;<OC4J>/xalan.jar;
<OC4J>/crimson.jar
```

where <OC4J> is the home folder of the application server installation.

**9** For **Sun J2EE** users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher. Open the Run-Time Settings dialog box and select the **Java VM** node. Add the following entries to the **Additional Classpath** section:

```
<J2EE>/j2ee.jar;<AppClientJar>
```

where <J2EE> is the home folder of the application server installation and <AppClientJar> is the full path to the application client jar file created automatically by the sdk tools during the deployment process.

**10** For **WebLogic 4.x - 5.x** Users:

Make sure that you have valid Java environment on your machine (path & classpath). Open the Run-Time Settings dialog box and select the **Java VM** node. Add the following two entries to the **Additional Classpath** section:

```
<WL>/classes;<WL>/lib/weblogicaux.jar
```

where <*WL*> is the home folder of the WebLogic installation.

**11** For **WebLogic 6.x** and **7.0** users:

Make sure that you have valid Java environment on your machine (path & classpath). WebLogic 6.1 requires JDK 1.3. Open the Run-Time Settings dialog box and select the **Java VM** node. Add the following entry to the **Additional Classpath** section:

```
<WL>/lib/weblogic.jar;    // Weblogic 6.x
<WL>/server/lib/weblogic.jar    // Weblogic 7.x
```

where *<WL>* is the home folder of the WebLogic installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

**12** Run the script. Click the **Run** button or 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 create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# Part X

## ERP/CRM Protocols

# 56

# Developing Web GUI-Level Scripts

This section provides information for creating Web GUI-level script for Oracle Web Applications 11i and PeopleSoft Enterprise users.

This chapter describes:

- About Developing Web GUI-Level Scripts
- Getting Started with Web GUI-Level Vusers
- Using GUI-Level Vuser Functions
- Understanding GUI-level Vuser Scripts
- Tips for Working with the GUI-Level Vusers

***The following information only applies to the Oracle Web Applications 11i and PeopleSoft Enterprise protocols.***

# About Developing Web GUI-Level Scripts

VuGen provides a solution for recording sessions on a Web GUI-level, specifically for Oracle Web Applications 11i and PeopleSoft Enterprise. These protocols emulate ERP/CRM enterprise tools that enable organizations to maintain company information and perform business processes from a single environment.

Oracle Web Applications 11i and PeopleSoft Enterprise provide Web access to all of the business processes. When you record a session, VuGen records all of the activity and creates a script. During replay, the script emulates the HTTP protocol communication between your browser and the server.

Many of the recorded pages contain non-HTML code such as Javascript. When recording in URL-based or HTML-based mode, VuGen included the Javascript as a sub-resource of the page's **web_url** function. In Web-GUI mode, VuGen creates an object oriented script that accurately interprets Javascript in the source code.

VuGen creates a GUI-level script that intuitively represents actions in the Web interface. For example, it generates a **web_button** function when you click a button to submit information, and generates a **web_edit_field** function when you enter text into an edit box.

# Getting Started with Web GUI-Level Vusers

This section provides an overview of the process of developing Oracle Web Applications 11i or PeopleSoft Enterprise scripts using VuGen. In addition to these steps, it is recommended that you review "Tips for Working with the GUI-Level Vusers" on page 838.

**To develop an Oracle Applications or PeopleSoft 8 script:**

**1 Record the actions using VuGen.**

Invoke VuGen and create a new file, multi-protocol or single, specifying Oracle Web Applications 11i or PeopleSoft Enterprise as the type from the ERP/CRM category. Record typical operations in your session. For details, see Chapter 4, "Recording with VuGen."

**2 Enhance the script.**

Enhance the script by inserting transactions, rendezvous points, and control-flow structures. For details, see Chapter 7, "Enhancing Vuser Scripts."

**3 Define parameters (optional).**

Define parameters for the recorded fixed-values. By substituting fixed-values with parameters, you can repeat the same business process many times using different values. For details, see Chapter 8, "Working with VuGen Parameters."

**4 Configure the run-time settings.**

The run-time settings control the script's behavior during test execution. These settings include the pacing, logging, think time, and connection information. For details, see Chapter 12, "Configuring Run-Time Settings."

**5 Save and run the script from VuGen.**

Save and run the script from VuGen to verify that it runs correctly.

For details, see "Tips for Working with the GUI-Level Vusers" on page 838, and Chapter 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Using GUI-Level Vuser Functions

During an Oracle Web Applications 11i or PeopleSoft 8 recording session, VuGen generates functions that emulate the communication between your browser and the server. The generated functions have a **web** prefix.

### Function List

The following is a list of functions that are specific for the GUI-Level recording. For complete syntax and information about other **web** functions, refer to the *Online Function Reference* (**Help > Function Reference**).

| Function Name | Description |
|---|---|
| **web_browser** | Performs an action on a browser. |
| **web_button** | Emulates a user clicking on button, submit, or reset input element. |
| **web_check_box** | Selects a check box. |
| **web_edit_field** | Enters data for text and password input types. |
| **web_dump_cache** | Saves the browser cache to a file. |
| **web_eval_java_script** | Evaluates a Javascript or DOM object. |
| **web_file** | Enters a path for file input types. |
| **web_image_link** | Emulates a user clicking on an image which is a hypertext link. |
| **web_image_submit** | Submits an image. |
| **web_list** | Selects an item from a list. |
| **web_load_cache** | Restores the browser cache from a file. |

| Function Name | Description |
|---|---|
| **web_map_area** | Activates an area within a client side map. |
| **web_radio_group** | Selects one button from a radio button group. |
| **web_reg_dialog** | Sets data for subsequent call to java script. |
| **web_static_image** | Emulates a user clicking on a static image. |
| **web_text_area** | Enters text in an input text area. |
| **web_text_link** | Emulates a user clicking on a hypertext link. |

### General API Notes

This section lists general notes about the GUI-level functions. Note that you can specify a regular expression for any object description, by preceding the text with "/RE" before the equals sign. For example:

```
web_text_link("Manage Assets",
        DESCRIPTION,
        "Text/RE=Manage Assets",
            LAST);
```

### Ordinals

The above functions use an **Ordinal** attribute. This attribute is a one-based index for the occurrence of the function with identical argument values. In the following example, the recorded **web_text_link** functions have identical arguments, except for the ordinal. The ordinal value of 2, indicates the second occurrence.

```
web_text_link("Manage Assets",
        DESCRIPTION,
        "Text=Manage Assets",
        "FrameName=main",
        LAST);

    web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Ordinal=2",
        "FrameName=main",
        LAST);
```

### Empty Strings

There is a difference between not specifying an argument and specifying it as an empty string. When you do not specify an argument, VuGen uses the default value or ignores it. When you list an argument, but assign it an empty string as a value, VuGen attempts to find a match with an empty string or no string at all. For example, omitting the **id** argument instructs VuGen to ignore the **id** property of the HTML element. Specifying "ID=" searches for HTML elements with no **id** property or with an empty ID.

### Browser/Frame Identification Arguments

- When recording with multiple browsers, you must specify a **BrowserTitle** or **BrowserOrdinal** argument.
- When there are multiple frames within the browser, you must specify a **FrameName** or **FrameHierarchyLevel** argument.
- Place all browser-related arguments at the end of the function's **DESCRIPTION** section. Place frame-related arguments immediately before the browser-related arguments, at the end of the **DESCRIPTION** section.

# Understanding GUI-level Vuser Scripts

GUI-level Vuser scripts typically contains several actions which make up a business process. By viewing the recorded functions, generated on a GUI level, you can determine the user's exact actions during the recording session.

For example, in a PeopleSoft Enterprise recording, the first stage contains the sign-in process. The browser opens on the PeopleSoft server start page and a user signs in by submitting a user name and password and clicking **Sign In**.

When you enter data in an edit field, VuGen generates a **web_edit_field** function. In the example that follows, the user entered VP5 into the **userid** text field, and a password into the **pwd** field which is encrypted.

```
vuser_init()
{
	web_url("about:blank",
		"URL=http://psft1/servlets/iclientservlet/peoplesoft8/?cmd=login",
		LAST);

	web_edit_field("userid",
		DESCRIPTION,
		"Type=text",
		"Name=userid",
		ACTION,
		"SetValue=VP5",
		LAST);

	web_edit_field("pwd",
		DESCRIPTION,
		"Type=password",
		"Name=pwd",
		ACTION,
		"SetEncryptedValue=3e52677bda7f4b",
		LAST);
…
```

When you click a button to submit data, VuGen generates **web_button**. (If the button is an image, VuGen generates **web_image_submit**.) In the following example, a user clicked the **Sign In** button.

```
…
      web_button("Sign In",
           DESCRIPTION,
           "Type=submit",
           "Tag=INPUT",
           "Value=Sign In",
           LAST);
      return 0;
}
```

The next section illustrates a typical action in which the user navigates to the **Asset ExpressAdd** process under the **Manage Assets** branch. The user navigates by clicking the text links of the desired branches, generating **web_text_link** functions.

```
web_text_link("Manage Assets_2",
           DESCRIPTION,
           "Text=Manage Assets",
           "Ordinal=2",
           "FrameName=main",
           LAST);

web_text_link("Use",
           DESCRIPTION,
           "Text=Use",
           "FrameName=main",
           LAST);

web_text_link("Asset ExpressAdd",
           DESCRIPTION,
           "Text=Asset ExpressAdd",
           "FrameName=main",
           LAST);
```

In the following section, the functions emulate typical user actions such as filling in fields and the selection of a list item.

```
web_edit_field("ASSET_DESCR",
        DESCRIPTION,
        "Type=text",
        "Name=ASSET_DESCR",
        "FrameName=main",
        ACTION,
        "SetValue=Car",
        LAST);
…
web_list("Year",
        DESCRIPTION,
        "Name=Year",
        "FrameName=CalFrame",
        ACTION,
        "Select=2000",
        LAST);
```

When you click on an image that is associated with an image map, VuGen generates a **web_map_area** function.

```
web_map_area("map2_2",
        DESCRIPTION,
        "MapName=map2",
        "Ordinal=20",
        "FrameName=CalFrame",
        LAST);
```

# Tips for Working with the GUI-Level Vusers

### Recording Tips

During recording, use only GUI objects that are within the browser's pane. Do not use any browser icons, controls, or menu items, such as Stop, Refresh, Home. You may, however, use the address bar and the Back and Forward buttons.

### 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.
- To improve Vuser performance, you can utilize VuGen's cache-simulating capabilities. For more information, see "Improving Performance Using Caching" on page 555.

# 57

# Setting Web GUI Recording Options

Before recording a script, you can set options to indicate the objects to be recorded and which properties should be included for each object.

This chapter describes:

- About Setting Web GUI Recording Options
- Configuring Object Identification
- Configuring Web Event Recording
- Selecting a Recording Level for GUI-Level Vusers

***The following information only applies to the Oracle Web Applications 11i and PeopleSoft Enterprise protocols.***

## About Setting Web GUI Recording Options

You can set common recording options in the following areas: General, Internet Protocol, and Network.

The following sections discusses the recording options that are specific for Oracle Applications and PeopleSoft Enterprise Vusers. For information on the other Recording Options, see the appropriate section:

- **General: Script**: see Chapter 5, "Setting Script Generation Preferences."
- **Network: Port Mapping**: see Chapter 6, "Configuring the Port Mappings."
- **Internet Protocol: Advanced**: see "Setting Advanced Recording Options," on page 564. Note that the **Save snapshot resources locally** and **Generate web_reg_find functions for page titles** options do not apply to GUI-based scripts (see explanation of GUI-based scripts below).

➤ **Internet Protocol: Correlation**: see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts." Note that there are built-in rules for the Oracle and PeopleSoft servers. To enable them, select the check box adjacent to the Oracle or PeopleSoft server name.

## Configuring Object Identification

When you record an operation on an object, VuGen learns a set of properties and values that uniquely describe the object within its parent object. In most cases, this description is sufficient to enable VuGen to identify the object during the run session.

If these mandatory property values are not sufficient to uniquely identify an object you record, VuGen can add some assistive properties and/or an ordinal identifier in order to create a unique description.

**Mandatory properties** are properties that VuGen always learns for a particular test object class.

**Assistive properties** are properties that VuGen learns only if the mandatory properties that VuGen learns for a particular object in your application are not sufficient to create a unique description. If several assistive properties are defined for an object class, then VuGen learns one assistive property at a time, and stops as soon as it creates a unique description for the object. If VuGen does learn assistive properties, those properties are added to the test object description.

You use the Object Identification dialog box (**Tools** > **Object Identification**) to configure the mandatory and assistive properties that VuGen uses to record descriptions of the objects in your application

The Object Identification dialog box also enables you to configure new user-defined classes and map them to an existing test object class so that VuGen can recognize objects from your user-defined classes when you run your test.

## Configuring Mandatory and Assistive Recording Properties

When you record an object of a specific class, VuGen learns several mandatory and assistive properties. If necessary, you can modify the properties that VuGen learns. For example:

- if you find that the description VuGen uses for a certain object class is not the most logical one for the objects in your application.
- if you expect that the values of the properties currently used in the object description may change.

During the run session, VuGen looks for objects that match all properties in the test object description—it does not distinguish between properties that were learned as mandatory properties and those that were learned as assistive properties.

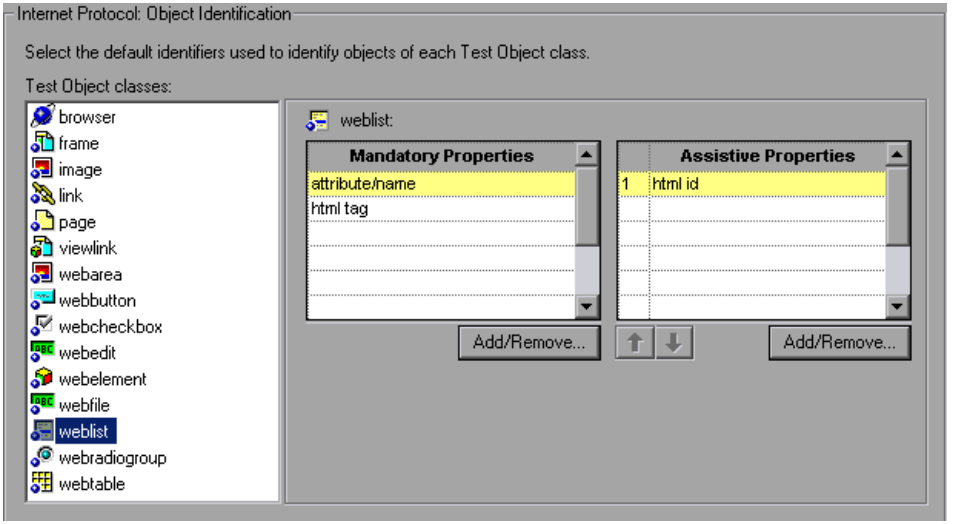
For example, the default mandatory properties for a Web Image object are the **alt**, **html tag**, and **image type** properties. There are no default assistive properties defined. Suppose your Web site contains several space holders for different collections of rotating advertisements. You want to record a test that clicks on the images in each one of these space holders. However, since each advertisement image has a different **alt** value, one alt value would be recorded when you create the test, and most likely another **alt** value will be captured when you run the test, causing the run to fail. In this case, you could remove the alt property from the Web Image mandatory properties list. Instead, since each ad image displayed in a certain space holder in your site has the same value for the image **name** property, you could add the **name** property to the mandatory properties to enable VuGen to uniquely identify the object.

Also, suppose that whenever a Web image is displayed more than once on a page (like a logo displayed on the bottom and top of a page), the Web designer adds a special **ID** property to the Image tag. Thus, the mandatory properties are sufficient to create a unique description for images that are only displayed once on the page, but you also want VuGen to learn the **ID** property for images that are displayed more than once on a page. To do this, you add the **ID** property as an assistive property, so that VuGen learns the **ID** property only when it is necessary for creating a unique test object description.

**To configure mandatory and assistive properties for a test object class:**

**1** Open the Recording Options dialog box. Choose **Tools** > **Recording Options**.

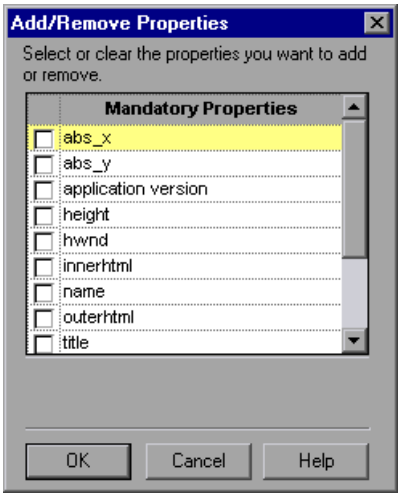
**2** Select the **Object Identification** node.




The common test object classes are displayed in the **Test object classes** list.

**3** In the **Test Object classes** list, select the test object class you want to configure.

**4** In the **Mandatory Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for mandatory properties opens.




**5** Select the properties you want to include in the Mandatory Properties list and/or clear the properties you want to remove from the list.

---

**Note:** You cannot include the same property in both the mandatory and assistive property lists.

---

You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

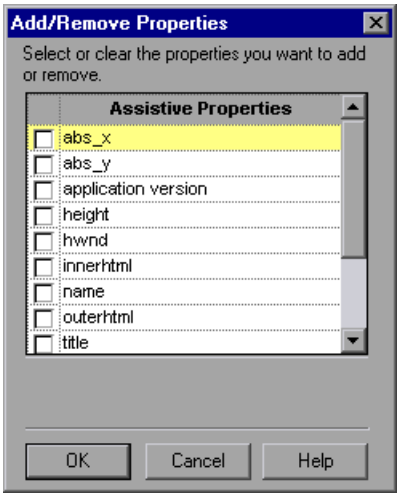
---

**Tip:** You can also add property names to the set of available properties for Web objects using the attribute/<PropertyName> notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property in the format attribute/<PropertyName> and click **OK**. The new property is added to the **Mandatory Properties** list. For example, to add a property called **MyColor**, enter attribute/MyColor.

---

**6** Click **OK** to close the Add/Remove Properties dialog box. The updated set of mandatory properties is displayed in the **Mandatory Properties** list.

**7** In the **Assistive Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for assistive properties opens.




**8** Select the properties you want to include in the assistive properties list and/or clear the properties you want to remove from the list.

---

**Note:** You cannot include the same property in both the mandatory and assistive property lists.

---

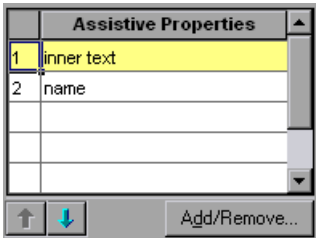
You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

---

**Tip:** You can also add property names to the set of available properties for Web objects using the attribute/<PropertyName> notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property in the format attribute/<PropertyName> and click **OK**. The new property is added to the **Assistive Properties** list. For example, to add a property called **MyColor**, enter attribute/MyColor.

---

**9** Click **OK** to close the Add/Remove Properties dialog box. The properties are displayed in the Assistive Properties list.






**10** Use the up and down arrows to set your preferred order for the assistive properties. When you record a test and assistive properties are necessary to create a unique object description, VuGen adds the assistive properties to the description one at a time until it has enough information to create a unique description, according to the order you set in the Assistive Properties list.

# Configuring Web Event Recording

VuGen creates your test by recording the events you perform on your Web-based application. An event is a notification that occurs in response to an operation, such as a change in state, or as a result of the user clicking the mouse or pressing a key while viewing the document.

You may find that you need to record more or fewer events than VuGen automatically records by default. You can modify the default event recording settings by using the Web Event Recording Configuration dialog box to select one of three standard configurations, or you can customize the individual event recording configuration settings to meet your specific needs.

This section describes how to configure VuGen's handling of Web Events:

- Selecting a Standard Event Recording Configuration
- Customizing the Event Recording Configuration
- Adding and Deleting Objects in the Custom Configuration Object List
- Adding and Deleting Listening Events for an Object
- Modifying the Listening and Recording Settings for an Event
- Saving and Loading Custom Event Configuration Files
- Resetting Event Recording Configuration Settings

For example, VuGen does not generally record mouseover events on link objects. If, however, you have a mouseover behavior connected to a link, it may be important for you to record the mouseover event. In this case, you could customize the configuration to record mouseover events on link objects whenever they are connected to a behavior.

---

**Notes:** Event configuration is a global setting and therefore affects all test that are recorded after you change the settings.

Changing the event configuration settings does not affect tests that have already been recorded. If you find that VuGen recorded more or less than

you need, change the event recording configuration and then re-record the part of your test that is affected by the change.

Changes to the custom Web event recording configuration settings do not take effect on open browsers. To apply your changes for an existing test, make the changes you need in the Web Event Recording Configuration dialog box, refresh any open browsers, and then start a new recording session.

---

## Selecting a Standard Event Recording Configuration

The Web Event Recording Configuration dialog box offers three standard event-configuration levels. By default, VuGen uses the **Basic** recording-configuration level. If VuGen does not record all the events you need, you may require a higher event-configuration level.

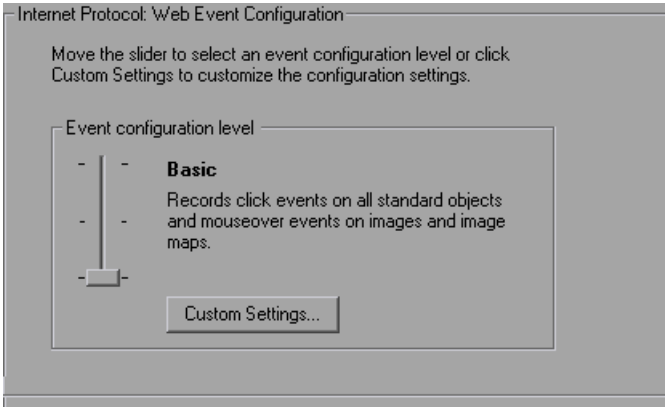
| Level | Description |
|---|---|
| **Basic** | **Default**<br>• Always records click events on standard Web objects such as images, buttons, and radio buttons.<br>• Always records the submit event within forms.<br>• Records click events on other objects with a handler or behavior connected.<br>For more information on handlers and behaviors, see "Listening Criteria" on page 854.<br>• Records the mouseover event on images and image maps only if the event following the mouseover is performed on the same object. |
| **Medium** | Records click events on the <DIV>, <SPAN>, and <TD> HTML tag objects, in addition to the objects recorded in the basic level. |

| Level | Description |
|---|---|
| **High** | Records mouseover, mousedown, and double-click events on objects with handlers or behaviors attached, in addition to the objects recorded in the basic level.<br>For more information on handlers and behaviors, see "Listening Criteria" on page 854. |

**To set a standard event-recording configuration:**

1 Open the Recording Options dialog box. Choose **Tools** > **Recording Options**.

2 Select the **Web Event Configuration** node.




3 Use the slider to select your preferred standard event recording configuration.

---

**Tip:** You can click the **Custom Settings** button to open the Custom Web Event Recording dialog box where you can customize the event recording configuration. For more information, see "Customizing the Event Recording Configuration," below.

---

4 Click **OK**.

## Customizing the Event Recording Configuration

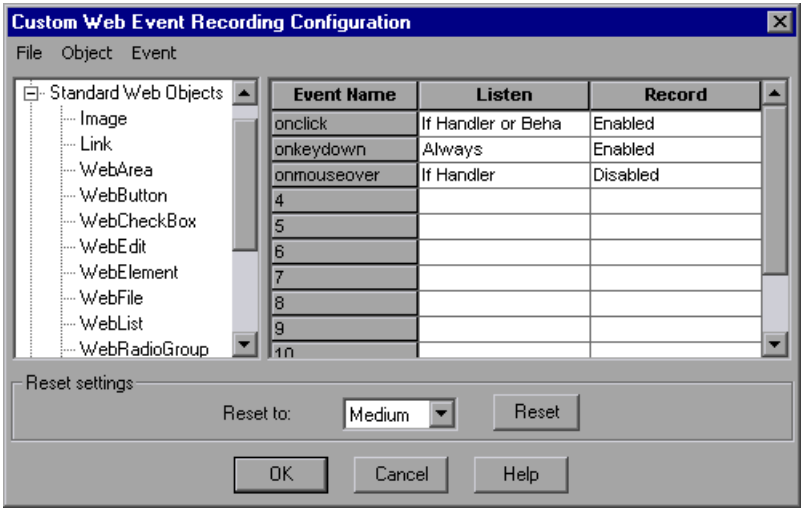
If the standard event configuration levels do not exactly match your recording needs, you can customize the event recording configuration using the Custom Web Event Recording Configuration dialog box.

The Custom Web Event Recording Configuration dialog box enables you to customize event recording in several ways. You can:

- add or delete objects to which VuGen should apply special listening or recording settings
- add or delete events for which VuGen should listen
- modify the listening or recording settings for an event

**To customize the event recording configuration:**

1. Open the Recording Options dialog box. Choose **Tools** > **Recording Options**.
2. Select the **Web Event Configuration** node.
3. Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.

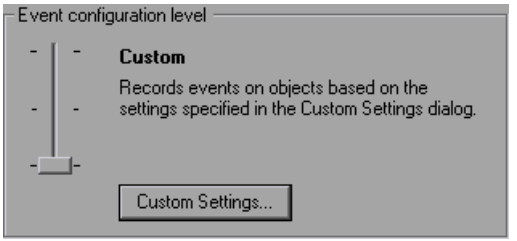


**4** Customize the event recording configuration using the following options:

| Option | Description |
|---|---|
| **Objects pane** | Displays a list of Web test object classes and HTML tag objects.<br>• To add an object, choose **Object** > **Add**.<br>• Only HTML Tag objects can be deleted. To delete an HTML object from the list, choose **Object** > **Delete**.<br>For more information, see "Adding and Deleting Objects in the Custom Configuration Object List" on page 851. |
| **Events pane** | Displays a list of events associated with the object.<br>• To add an event to the Events pane, choose **Event** > **Add**.<br>• To delete an event, choose **Event** > **Delete**.<br>For more information, see "Adding and Deleting Listening Events for an Object" on page 853. |
| **Event Name** | The name of the event. |
| **Listen** | The criteria for when VuGen listens to the event.<br>• **Always**—Always listens to the event.<br>• **If Handler**—Listens to the event if a handler is attached to it. A handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs.<br>• **If Behavior**—Listens to the event if a DHTML behavior is attached to it. A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior.<br>• **If Handler or Behavior**—Listens to the event if a handler or behavior is attached to it.<br>• **Never**—Never listens to the event.<br>For more information, see "Modifying the Listening and Recording Settings for an Event" on page 854. |

| Option | Description |
|---|---|
| **Record** | Enables or disables recording of the event for the selected object, or enables recording of the event only if the subsequent event occurs on the same object. |
| **Reset** | Enables you to reset your settings to a preconfigured level. |

**5** Click **OK**. The Event Configuration Level displays **Custom**.




### Adding and Deleting Objects in the Custom Configuration Object List

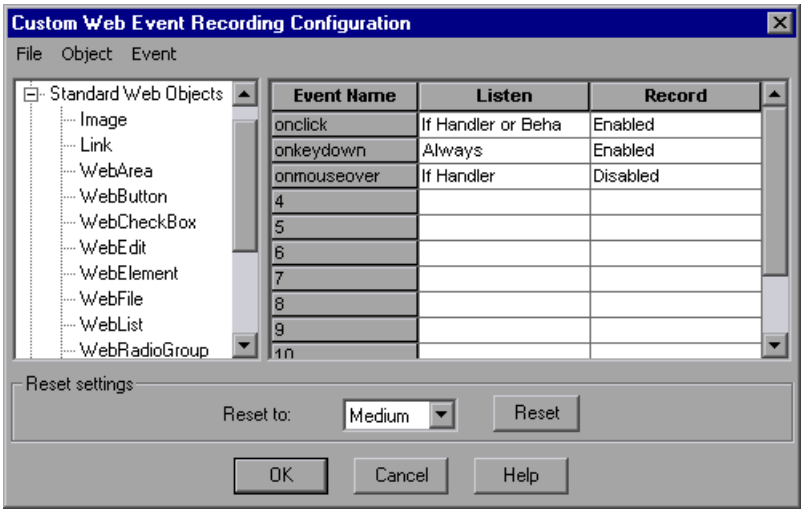
The Custom Web Event Recording Configuration dialog box lists objects in an object hierarchy. The top of the hierarchy is **Any Web Object**. The settings for Any Web Object apply to any object on the Web page being tested, for which there is no specific event recording configuration set. Below this are the **Web Objects** and **HTML Tag Objects** categories, each of which contains a list of objects.

When working with the objects in the Custom Web Event Recording Configuration dialog box, keep the following principles in mind:

- If an object is listed in the Custom Web Event Recording Configuration dialog box, then the settings for that object override the settings for Any Web Object.
- You cannot delete or add to the list of objects in the **Web Objects** category, but you can modify the settings for any of these objects.
- You can add any HTML Tag object in your Web page to the HTML Tag Objects category.

**To add objects to the event configuration object list:**

1 In the Custom Web Event Recording Configuration dialog box, choose **Object** > **Add**. A **New Object** object is displayed in the HTML Tag Objects list. A **New Object** object is displayed in the HTML Tag Objects list.




2 Click **New Object** to rename it. Enter the exact HTML Tag name.

By default the new object is set to listen and record **onclick** events with handlers attached.

For more information on adding or deleting events, see "Adding and Deleting Listening Events for an Object," below. For more information on listening and recording settings, see "Modifying the Listening and Recording Settings for an Event" on page 854.

**To delete objects from the HTML Tag Objects list:**

1 From the Custom Web Event Recording Configuration dialog box, select the object in the HTML Tag Objects category that you want to delete.

2 Choose **Object** > **Delete**. The object is deleted from the list.

**Note:** You cannot delete objects from the **Web Objects** category.

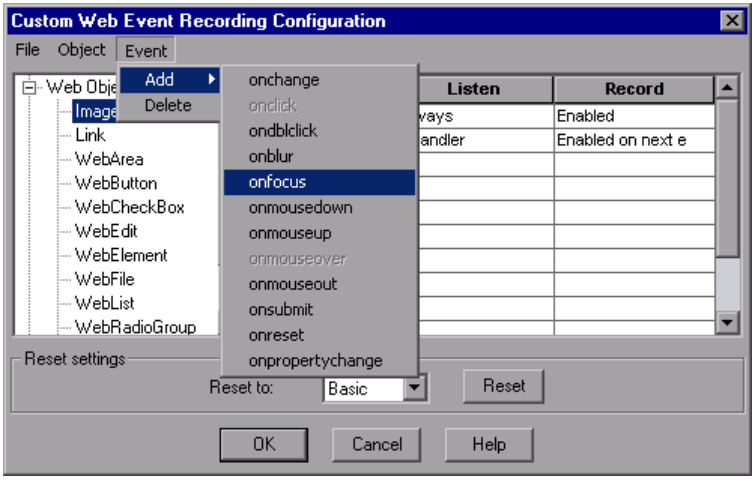
### Adding and Deleting Listening Events for an Object

You can modify the list of events that trigger VuGen to listen to an object.

**To add listening events for an object:**

1 In the Custom Web Event Recording Configuration dialog box, select the object to which you want to add an event, or select **Any Web Object**.

2 Choose **Event** > **Add**. A list of available events opens.




3 Select the event you want to add. The event is displayed in the Event Name column in alphabetical order. By default, VuGen listens to the event when a handler is attached and always records the event (as long as it is listened to at some level).

For more information on listening and recording settings, see “Modifying the Listening and Recording Settings for an Event,” below.

**To delete listening events for an object:**

1 In the Custom Web Event Recording Configuration dialog box, select the object from which you want delete an event, or select **Any Web Object**.

2 Select the event you want to delete from the **Event Name** column.

3 Choose **Event** > **Delete**. The event is deleted from the **Event Name** column.

## Modifying the Listening and Recording Settings for an Event

You can select the listening criteria and set the recording status for each event listed for each object.

---

**Note:** The listen and record settings are mutually independent. This means that you can choose to listen to an event for particular object, but not record it, or you can choose not to listen to an event for an object, but still record the event. For more information, see "Tips for Working with Event Listening and Recording" on page 856.

---

### Listening Criteria

For each event, you can instruct VuGen to listen every time the event occurs on the object if an event handler is attached to the event, if a DHTML behavior is attached to the event, if an event handler or DHTML behavior are attached to the event, or to never listen to the event.

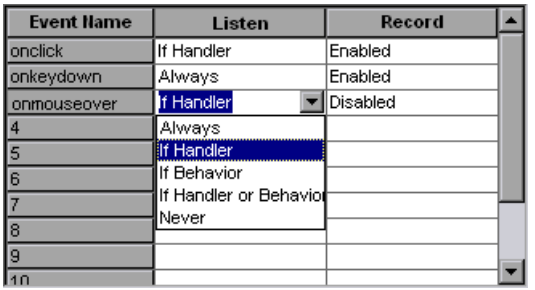
An event **handler** is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs.

A DHTML **behavior** encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior.

**To specify the listening criterion for an event:**

1 From the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the listening criterion or select **Any Web Object**.

**2** In the row of the event you want to modify, select the listening criterion you want from the **Listen** column.




You can select **Always, If Handler**, **If Behavior**, **If Handler or Behavior**, or **Never**.

### Recording Status

For each event, you can enable recording, disable recording, or enable recording only if the next event is dependent on the selected event.

- **Enabled**—Records the event each time it occurs on the object as long as VuGen listens to the event on the selected object, or on another object to which the event bubbles.

  **Bubbling** is the process whereby, when an event occurs on a child object, the event can travel up the chain of hierarchy within the HTML code until it encounters an event handler to process the event.

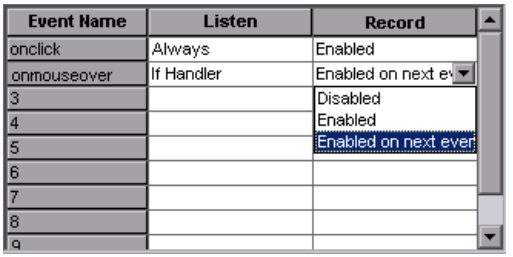
- **Disabled**—Does not record the specified event and ignores event bubbling where applicable.

- **Enabled on next event**—Same as **Enabled**, except that it records the event only if the subsequent event occurs on the same object. For example, suppose a mouseover behavior modifies an image link. You may not want to record the mouseover event each time you happen to move the mouse over this image. Because only the image that is displayed after the mouseover event enables the link event, however, it is essential that the mouseover event is recorded before a click event on the same object. This option applies only to the Image and WebArea objects.

**To set the recording status for an event:**

1 From the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the recording status or select **Any Web Object**.

2 In the row of the event you want to modify, select a recording status from the Record column.




### Tips for Working with Event Listening and Recording

It can sometimes be difficult to find the ideal listen and recording settings. When defining these settings, keep in mind the following guidelines:

- If settings for different objects in the Objects Pane conflict, VuGen gives first priority to settings for specific **HTML Tag Objects** and second priority to **Web Objects** settings. VuGen only applies the settings for **Any Web Object** to Web objects that were not defined in the **HTML Tag Object** or **Web Objects** areas.
- To record an event on an object, you must instruct VuGen to listen for the event, and to record the event when it occurs. You can listen for an event on a child object, even if a parent object contains the handler or behavior, or you can listen for an event on a parent object, even if the child object contains the handler or behavior.

  However, you must enable recording for the event on the source object (the one on which the event actually occurs, regardless of which parent object contains the handler or behavior).

  For example, suppose a table cell with an **onmouseover** event handler contains two images. When a user touches either of the images with the mouse pointer, the event also bubbles up to the cell, and the bubbling includes information on which image was actually touched. You can record this mouseover event by:

- Setting **Listen** on the <TD> tag mouseover event to **If Handler** (so that VuGen "hears" the event when it occurs), while disabling recording on it, and then setting **Listen** on the <IMG> tag mouseover event to **Never**, while setting **Record** on the <IMG> tag to **Enable** (to record the mouseover event on the image after it is listened to at the <TD> level).
- Setting **Listen** on the <IMG> tag mouseover event to **Always** (to listen for the mouseover event even though the image tag does not contain a behavior or handler), and setting **Record** on the <IMG> tag to **Enabled** (to record the mouseover event on the image).

- Instructing VuGen to listen for many events on many objects may lower performance, so try to limit listening settings to the required objects.
- In rare situations, listening to the object on which the event occurs (the source object) can interfere with the event.

  If you find that your application works properly until you begin recording on the application using VuGen, your listen settings may be interfering.

  If this problem occurs with a mouse event, try selecting the appropriate **Use standard Windows mouse events** option(s) in the Advanced Web Options dialog box.

  If this problem occurs with a keyboard or internal event, or the **Use standard Windows mouse events** option does not solve your problem, set the listen settings for the event to **Never** on the source object (but keep the record setting enabled on the source object), and set the listen settings to **Always** for a parent object.

### Saving and Loading Custom Event Configuration Files

You can save the changes you make in the Custom Web Event Recording Configuration dialog box, and load them at any time.

**To save a custom configuration:**

1. Customize the event recording configuration as desired. For more information on how to customize the configuration, see "Customizing the Event Recording Configuration" on page 849.
2. In the Custom Web Event Recording Configuration dialog box, Choose **File > Save Configuration As**. The Save As dialog box opens.

3 Navigate to the folder in which you want to save your event configuration file and enter a configuration file name. The extension for configuration files is **.xml**.

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

**To load a custom configuration:**

1 Choose **Tools** > **Web Event Recording Configuration** and then click **Custom Settings** to open the Custom Web Event Recording Configuration dialog box.

2 Choose **File** > **Load Configuration**. The Open dialog box opens.

3 Locate the event configuration file (**.xml**) that you want to load and click **Open**. The dialog box closes and the selected configuration is loaded.

### Resetting Event Recording Configuration Settings

You can restore standard settings after you set Custom settings by resetting the event recording configuration settings to the basic level from the Web Event Recording Configuration dialog box.

---

**Note:** When you choose to reset standard settings, your custom settings are cleared completely. If you do not want to lose your changes, be sure to save your settings in an event configuration file. For more information, see "Saving and Loading Custom Event Configuration Files" on page 857.

---

**To reset basic level configuration settings from the Web Event Recording Configuration dialog box:**

1 Choose **Tools** > **Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.

2 Click **Default**. The standard configuration slider is displayed again and all event settings are restored to the **Basic** event recording configuration level.

3 If you want to select a different standard configuration level, see "Selecting a Standard Event Recording Configuration" on page 847.

You can also restore the settings to a specific (base) custom configuration from within the Custom Web Event Recording Configuration dialog box so that you can begin customizing from that point.

**To reset the settings to a custom level from the Custom Web Event Recording Configuration dialog box:**

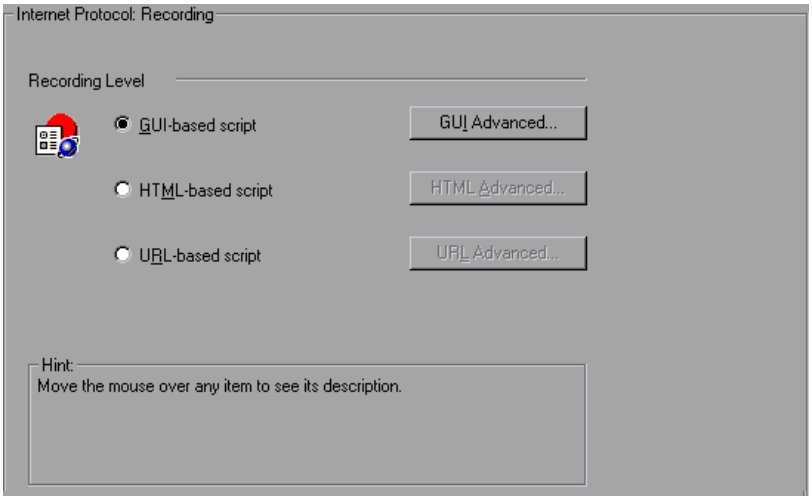
1. Choose **Tools** > **Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.
2. Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.
3. In the **Reset to** box, select the standard event recording level you want.
4. Click **Reset**. All event settings are restored to the defaults for the level you selected.

## Selecting a Recording Level for GUI-Level Vusers

VuGen lets you specify what information to record and which functions to use when generating a script by selecting a recording level. The recording level you select, depends on your needs and environment. The available levels are **GUI-based script**, **HTML-based script,** and **URL-based script**. Follow these guidelines in deciding which recording level to choose.

- For browser applications with JavaScript, use the GUI-based level. This is the ideal recording level for Oracle Web Applications 11i and PeopleSoft Enterprise.
- For browser applications without JavaScript, use the HTML-based level.

➤ For non-browser applications, use the URL-based level.




The **GUI-based script** level instructs VuGen to record HTML actions as context sensitive GUI functions such as **web_text_link**.

You can also indicate the action to take if VuGen cannot successfully replay the GUI-based script—generate the GUI-based script anyway, or fallback to an HTML-based script—by clearing or selecting the **Use HTML-based script as fallback** option. For more information, see "Setting Advanced GUI-Based Options," on page 863.

```
/* GUI-based mode - CS type functions with JavaScript support*//
vuser_init()
{
	web_url("about:blank",
		"URL=http://psft1/servlets/iclientservlet/peoplesoft8/?cmd=login",
		LAST);

	web_edit_field("userid",
		DESCRIPTION,
		"Type=text",
		"Name=userid",
		ACTION,
		"SetValue=VP5",
		LAST);
…
```

The **HTML-based script** level generates a separate step for each HTML user action. The steps are also intuitive, but they do not reflect true emulation of the JavaScript code.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("MercuryWebTours",
        "URL=http://localhost/MercuryWebTours/",
        "TargetFrame=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

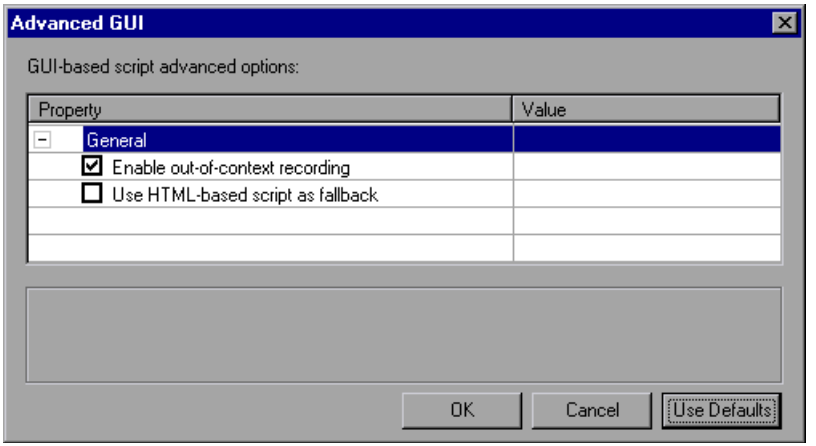
web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
        "Snapshot=t4.inf",
        LAST);

web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
…
```

For additional information about the different recording levels and their options, see Chapter 41, “Setting Recording Options for Web Vusers.”

## Setting Advanced GUI-Based Options

VuGen lets you set advanced options for GUI-based level recording:




**Enable out-of-context recording:** VuGen does not natively support the recording of ActiveX controls and Java applets. You can instruct VuGen to create a URL-based script for ActiveX controls and Java applets, so that they will be replayed. Since these functions are not part of the native recording, they are referred to as out-of-context recording. (enabled by default)

**Use HTML-based script as fallback:** You can indicate what action VuGen should take if it cannot successfully replay the GUI-based script—generate the GUI-based script anyway, or fallback to an HTML-based script. (disabled by default)

In the following example, the script was regenerated with the out-of-context recording option enabled.

```
web_text_link("Dialog",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Text=Dialog",
        "Ordinal=1",
        LAST);

web_url("DialogApplet.class",
    "URL=http://localhost/Java/java/awt/dialog/DialogApplet.class",
    "Resource=1",
    "RecContentType=application/octet-stream",
    "Referer=",
    LAST);

web_url("DialogDemo_Panel.class",
    "URL=http://localhost/Java/java/awt/dialog/DialogDemo_Panel.class",
    "Resource=1",
    "RecContentType=application/octet-stream",
    "Referer=",
    LAST);

web_url("DialogDemo_Panel$SymItem.class",
    "URL=http://localhost/Java/java/awt/dialog/DialogDemo_Panel$Sym-
Item.class",
    "Resource=1",
    "RecContentType=application/octet-stream",
    "Referer=",
    LAST);
. . .
```

If you disable this option, VuGen does not generate code for the ActiveX controls and Java applets as illustrated in the following example.

It only generated the **web_text_link** function—not the **web_url** functions containing the class files.

```
web_text_link("Dialog",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Text=Dialog",
        "Ordinal=1",
        LAST);
```

# 58

# 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:

- About Creating Oracle NCA Vuser Scripts
- Getting Started with Oracle NCA Vusers
- Recording Guidelines
- Enabling the Recording of Objects by Name
- Oracle Applications via the Personal Home Page
- Using Oracle NCA Vuser Functions
- Understanding Oracle NCA Vusers
- Configuring Oracle NCA Run-Time Settings
- Testing Oracle NCA Applications
- Correlating Oracle NCA Statements for Load Balancing
- Additional Recommended Correlations
- Recording in Pragma Mode

***The following information applies only to the Oracle NCA protocol.***

# About Creating Oracle NCA Vuser Scripts

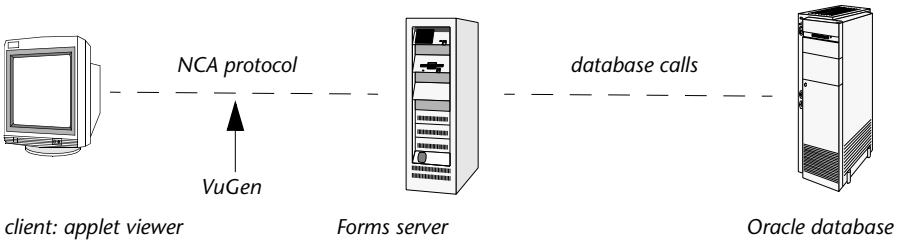
Oracle NCA is a Java-based database protocol. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer.

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 Script dialog box. For more information, see Chapter 4, "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 VuGen supports your version of Oracle Forms. For more information, see "Recording Guidelines" on page 870, and the Readme file.

**2 Create a skeleton Oracle NCA Vuser script.**

Use VuGen to create a skeleton Oracle NCA Vuser script. For details, see Chapter 4, "Recording with VuGen."

**3 Record typical user actions.**

Begin recording, and perform typical actions and business processes from the applet viewer. VuGen records your actions and generates a Vuser script. For details, see Chapter 4, "Recording with VuGen."

**4 Enhance the Vuser script.**

Use the Insert menu to add transactions, rendezvous points, comments, and messages in order to enhance the Vuser script. For details, see Chapter 7, "Enhancing Vuser Scripts."

**5 Parameterize the script.**

Replace recorded constants with parameters. For details, see Chapter 8, "Working with VuGen Parameters."

**6 Set the run-time properties for the script.**

Configure run-time settings for the Vuser script. The run-time settings define certain aspects of the script execution. For details, see Chapter 12, "Configuring Run-Time Settings."

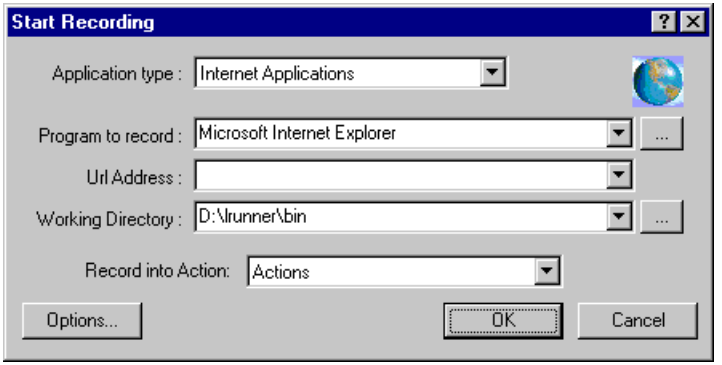
**7 Save and run the Vuser script.**

Run the script from VuGen and view the execution log for run-time information. For details, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

## Recording Guidelines

When recording an Oracle NCA Vuser script, follow these guidelines:

➤ Specify which browser VuGen should use when recording an Oracle NCA session. In the Start Recording dialog box, select the desired browser in the **Program to Record** list. The list contains all of the available browsers.




➤ Close all browsers before you begin recording.

- Record the login procedure in the **vuser_init** section. Record a typical business process in the **Actions** section. When you run the script, you can then specify multiple iterations for a specific business process. For more information, see "Creating New Virtual User Scripts" on page 56.

```
vuser_init()
{
connect_server("199.203.78.170", "9000"/*version=107*/,"
      module=e:\\appsnca\\fnd\\7.5\\forms\\us\\fndscsgn
      userid=applsyspub/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 values.

## Enabling the Recording of Objects by Name

When recording an Oracle NCA script, you must record the session using object names instead of the standard object ID. If the script is recorded using the object ID, replay will fail because the ID is generated dynamically by the server and differs between record and replay. You can verify that your script is being recorded with object names by examining the **nca_connect_server** statement.

```
nca_connect_server("199.35.107.119","9002"/*version=11i*/,"module=/d1/oracle
/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
- Forms Configuration File

Note that the ability to capture the developer name for all objects was introduced in Oracle Forms6i Patch 9 (Oracle Forms Version: 6.0.8.18.3). Test Starter Kit scripts that were written before the release of Oracle Forms 6i Patch 9 will not have the developer name as part of an object's physical description, except for the edit fields.

### Startup HTML File

If you have access to the startup HTML file, you instruct VuGen to record object names instead of its object ID by setting the **record=names** flag in the startup file, the file that is loaded when you start the Oracle NCA application.

Edit the startup file that is called when the applet viewer begins. Modify the line:

```
<PARAM name="serverArgs … fndnam=APPS">
```

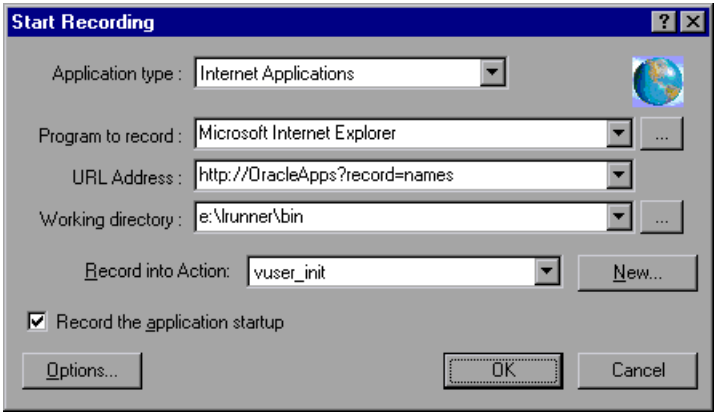
and add the Oracle key "record=names":

```
<PARAM name="serverArgs … fndnam=APPS record=names">
```

### URL to Record

If you do not have access to the startup HTML file, you can still have Oracle NCA record object names instead of its object ID by modifying the URL to record. The following solution only works if the startup HTML file does not reference another file while loading.

For this solution, you add "**?record=names**" after the URL in the Start Recording dialog box, after the URL name to record. This allows VuGen to record object names for the session.




## Forms Configuration File

If the application has a startup HTML file that references a Forms Web CGI configuration file **formsweb.cfg** (a common reference), you may encounter problems if you add **record=names** to the Startup file.

In this situation, you should modify the configuration file.

**To modify the configuration file to record object names:**

1. Go to the Forms Web CGI configuration file.
2. Define a new parameter in this file (see sample Web CGI configuration file below for this change).

```
serverApp=forecast
serverPort=9001
serverHost=easgdev1.dats.ml.com
connectMode=socket
archive=f60web.jar
archive_ie=f60all.cab
xrecord=names
```

3. Open the startup HTML file and locate PARAM NAME="serverArgs".

**4** Add the variable name as an argument to the ServerArgs parameter, for example, **record=%xrecord%.**

```
<PARAM NAME="serverArgs" VALUE="module=%form% userid=%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 basejinin.hmt file add the following line to the parameter definitions:

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the <EMBED> tag, add the following line:

```
...
serverApp="%serverApp%"
logo="%logo%"
imageBase="%imageBase%"
formsMessageListener="%formsMessageListener%"
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file **formsweb.cfg**, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the **basejini.htm** file and store it at another location. In the servlet configuration file, edit the **baseHTMLJinitiator** parameter to point to the new file.

# Oracle Applications via the Personal Home Page

When launching Oracle Forms 6i applications by logging in through the **Personal Home Page,** you must set several system profile options at the user level. It is desirable to pass such variables at the user level, and not at the site level, where it will affect all users.

**To configure the "ICX: Forms Launcher" profile:**

1. Sign on to the application and select the "System Administrator" responsibility.
2. Select **Profile/System** from the Navigator menu.
3. Within the **Find System Profile Values** form:

   Select the **Display:Site** option

   **Users** = <your user logon> (i.e. operations, mfg, and so on)

   **Enter Profile** =%ICX%Launch%

   Click **Find**.
4. Update the User value to the **ICX:Forms Launcher** profile:

   If no parameter has been passed to the URL, append the following string to the end of the URL of the user value: **?**play=&record=names

   If a parameter has been passed to the URL, append the following string to the end of the URL of the user value: **&**play=&record=names
5. Save the transaction.
6. Log out of the Oracle Forms session.
7. Log out of the Personal Home Page session.
8. Sign on again via the **Personal Home Page** using your username.

If you were unable to update the ICX: Forms Launcher profile option at the user level, open the **Application Developer** responsibility and select the **Updatable** option for the ICX_FORMS_LAUNCHER profile.

The first parameter passed to the URL, must begin with a question mark (?). You pass all subsequent parameters with an ampersand (&). In most cases, the URL already contains parameters, which you can identify by searching for a question mark.

## Using Oracle NCA Vuser Functions

VuGen 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

| | |
|---|---|
| **nca_button_double_press** | Performs a double press on a push button. |
| **nca_button_press** | Activates a push button. |
| **nca_button_set** | Sets the state of the specified button. |

### Combo Box Object Functions

| | |
|---|---|
| **nca_combo_select_item** | Selects an item in a combo box. |
| **nca_combo_set_item** | Sets a new item in a combo box. |

### Connection Functions

| | |
|---|---|
| **nca_connect_server** | Connects to an Oracle NCA server. |
| **nca_logon_connect** | Performs a login to an Oracle NCA database. |
| **nca_logon_cancel** | Disconnects from an Oracle NCA database. |

### Edit Object Functions

| | |
|---|---|
| **nca_edit_box_press** | Clicks on an edit box message. |
| **nca_edit_click** | Clicks in an edit object. |
| **nca_edit_get_text** | Returns the text in an edit object. |
| **nca_edit_press** | Activates the browse button in an edit field. |
| **nca_edit_set** | Replaces the entire contents of an edit object. |

### Flex Object Functions

| | |
|---|---|
| **nca_flex_click_cell** | Clicks a table cell in a Flexfield window. |
| **nca_flex_get_cell_data** | Gets data from a Flexfield cell. |
| **nca_flex_get_column_name** | Gets the name of a column in a Flexfield window. |
| **nca_flex_press_clear** | Clicks **Clear** in a Flexfield window. |
| **nca_flex_press_find** | Clicks **Find** in a Flexfield window. |
| **nca_flex_press_help** | Clicks **Help** in a Flexfield window. |
| **nca_flex_press_lov** | Clicks on the List of Values button in a Flexfield window. |
| **nca_flex_press_ok** | Clicks **OK** in a Flexfield window. |
| **nca_flex_set_cell_data** | Inserts data in a Flexfield window. |
| **nca_flex_set_cell_data_press_ok** | Clicks **OK** in a Flexfield window after data is entered. |

### List Item Functions

| | |
|---|---|
| **nca_list_activate_item** | Activates an item in a list (double-click). |
| **nca_list_select_index_item** | Selects a list item by its index. |
| **nca_list_select_item** | Selects a list item by its name. |
| **nca_lov_auto_select** | Specifies the first letter of an item. |
| **nca_lov_find_value** | Clicks **Find** in a List of Values window. |
| **nca_lov_get_item_name** | Retrieves the name of an entry in a list of values by the entry's index number. |
| **nca_lov_retrieve_items** | Retrieves a list of values. |
| **nca_lov_select_index_item** | Selects an item from a list of values by its index number. |
| **nca_lov_select_item** | Selects an item from a list of values. |
| **nca_lov_select_random_item** | Selects a random item from a list of values. |

### Java Object Functions

| | |
|---|---|
| **nca_java_action** | Performs an event on a Java object. |
| **nca_java_get_value** | Retrieves the value of a Java object. |
| **nca_java_set_reply_property** | Sets a reply property for a Java object. |

### Menu Object Functions

| | |
|---|---|
| **nca_menu_select_item** | Selects an item from a menu. |

### Message Functions

| | |
|---|---|
| **nca_popup_message_press** | Clicks a button in a popup window. |
| **nca_message_box_press** | Clicks a button in a message window. |

### Object Functions

| | |
|---|---|
| **nca_obj_get_info** | Returns the value of an object property. |
| **nca_obj_mouse_click** | Clicks within an object. |
| **nca_obj_mouse_dbl_click** | Double-clicks within an object. |
| **nca_obj_status** | Returns the status of the specified object. |
| **nca_obj_type** | Types special characters into an edit box. |

### Response Object Functions

| | |
|---|---|
| **nca_response_press_lov** | Clicks a drop down arrow in a Response box. |
| **nca_response_press_ok** | Clicks **OK** inside a Response box. |
| **nca_response_set_cell_data** | Inserts data into a cell in a Response box. |
| **nca_response_set_data** | Inserts data into a Response box. |

### Scroll Object Functions

| | |
|---|---|
| **nca_scroll_drag_from_min** | Drags the scroll to the specified distance from the minimum position (0). |
| **nca_scroll_line** | Scrolls the specified number of lines. |
| **nca_scroll_page** | Scrolls the specified number of pages. |

### Session Functions

| | |
|---|---|
| **nca_console_get_text** | Retrieves the console message. |
| **nca_set_iteration_offset** | Sets an offset value for an object ID. |
| **nca_set_server_response_time** | Sets the server response time. |
| **nca_set_exception** | Specifies how to handle exceptions. |
| **nca_set_think_time** | Sets the think time range. |

**Tree Object Functions**

| | |
|---|---|
| **nca_tree_activate_item** | Activates an item in an NCA tree. |
| **nca_tree_collapse_item** | Collapses a tree item. |
| **nca_tree_expand_item** | Expands a tree item. |
| **nca_tree_select_item** | Selects an item in an NCA tree. |

**Window Object Functions**

| | |
|---|---|
| **nca_win_get_info** | Returns the value of an window property. |
| **nca_win_close** | Closes a window. |
| **nca_set_window** | 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 7, "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 7, "Enhancing Vuser Scripts." For a list of the available Oracle NCA Vuser functions, see "Using Oracle NCA Vuser Functions" on page 877. 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://usscifforms05.sfb.na/servlet/f60servlet\?config
       =mult", LAST);
```

In the above example, the right boundary is \r. The actual right boundary may differ between systems.

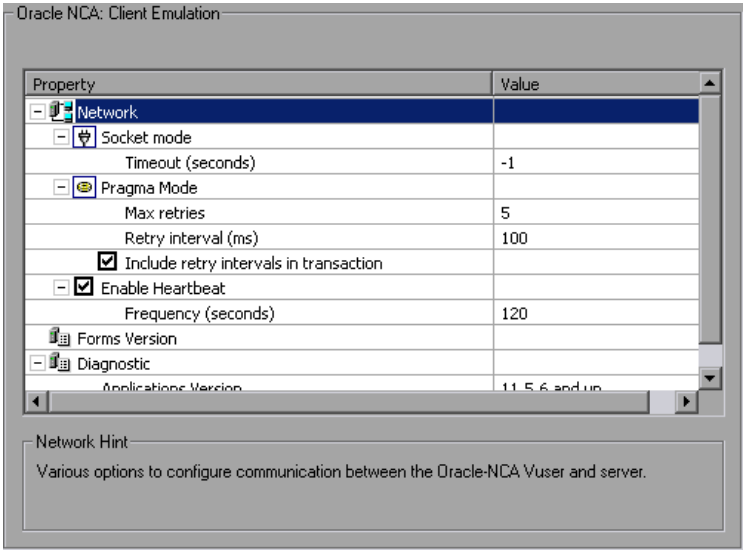
# Configuring Oracle NCA Run-Time Settings

Before running your script, you can set the run-time settings to allow the script to accurately emulate a real user. For information on the general run-time settings for all protocols, such as think time, pacing, and logging, see Chapter 12, "Configuring Run-Time Settings." For network speed related settings, see Chapter 13, "Configuring Network Run-Time Settings."

The following section describes the run-time settings specific to Oracle NCA Vusers. These run-time setting allow you to indicate the communication parameters.

## Configuring Oracle NCA Client Emulation Run-Time Settings

You can configure several network settings to accurately emulate an Oracle NCA client.




You can set the following options:

### Socket Mode

The communication to and from the client is performed on a socket level—not on the higher HTTP level.

**Timeout (seconds):** The time that an Oracle NCA Vuser waits for a response from the server. The default value of -1 disables the timeout and the client waits indefinitely.

### Pragma Mode

In Pragma mode, communication is carried out in the Oracle-defined Pragma mode. This communication level, above the HTTP and Servlet levels, is characterized by the periodic sending of messages. In this mode, the client recognizes that the server cannot respond with data immediately. The server sends messages at given intervals until it is able to send the requested data.

- **Max Retries**: indicates the maximum number of **IfError** messages the client will accept from the server before issuing an error. **IfError** messages are the periodic messages the server sends to the client, indicating that it will respond with the data as soon as it is able.
- **Retry Interval** defines the interval between retries in the case of **IfError** messages.
- **Include retry intervals in transaction:** includes the interval between retires time, as part of the transaction duration time.

For information about recording in Pragma mode, see "Recording in Pragma Mode" on page 893.

### Heartbeat

You can enable or disable the heartbeat sent to the Oracle server. The heartbeat verifies that there is proper communication with the server. If you are experiencing a heavy load on the Oracle NCA server, disable the heartbeat. If you enable the heartbeat, you can set the frequency of how often heartbeat messages are sent to the server.

**Enable Heartbeat:** By default, a heartbeat signal is sent to the server. To disable it, clear the checkbox.

**Frequency:** The frequency of the heartbeat signal. The default is 120 seconds.

### Forms

You can specify the version of the Oracle Forms server detected during recording.

**Version:** Modify this setting only if the server was upgraded since the recording.

### Diagnostic

This section lets you provide information about diagnostic modules for the database layer of Oracle Applications.

**Application version:** The version of Oracle Application. This option is relevant when using Oracle Application—not a custom Oracle NCA application. It is only required when using Oracle database breakdown.

**To set the Client Emulation settings:**

1. Open the Run-Time Settings dialog box. Choose **Vuser** > **Run-Time Settings** or click the **Run-Time Settings** button on the VuGen toolbar.



2. Select the **Oracle NCA:Client Emulation** node from the Run-Time settings tree.
3. Set the network timeout value in seconds. To instruct the client to wait indefinitely for a server response, use the default value of -1.
4. When working in Pragma mode, specify the number of retries **Max Retries**, (**IfError** messages) for the client to accept before issuing an error. The default is 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 6, "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.

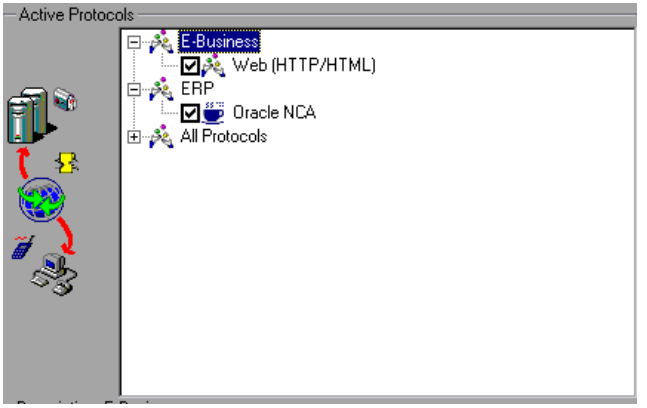
```
web_set_sockets_option("SSL_VERSION","3");
```

## 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 Script**, and select the Web protocol in the Protocols section.

## Determining the Recording Mode

When recording Oracle NCA scripts: VuGen automatically determines the correct connection mode: HTTP or Socket mode. Generally, you are not required to modify any of the recording settings as VuGen auto-detects the system configuration. In systems where the standard port mapping are reserved by other applications, you may need to modify the Port Mapping settings, depending on the recording mode.

You can determine the recording mode in one of the following ways:

➤ When using the NCA application, open the Java Console.

```
proxyHost=null
proxyPort=0
connectMode=HTTP
Forms Applet version is: 60812
```

The **connectMode** entry indicates **HTTP**, **HTTPS**, or **socket**.

➤ After recording an NCA session, open the **default.cfg** file in the Vuser directory and check the value of the **UseHttpConnectMode** entry.

```
[HttpConnectMode]
UseHttpConnectMode= 2
// 0 = socket 1 = http 2 = https
```

When defining a new port mapping int he Server Entry dialog box, use a **Service ID** of HTTP for HTTP or HTTPS modes. For Socket mode, use a **Service ID** of NCA.

For more information about Port Mapping settings, see Chapter 6, "Configuring the Port Mappings."

### Recording Trace Information for Oracle DB

To debug your script, you can use the Oracle DB breakdown graphs. To gather data for this graph, you turn on the trace mechanism before running the script.

To manually turn on the tracing mechanism, use the **nca_set_custom_dbtrace** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

# Correlating Oracle NCA Statements for Load Balancing

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

**To correlate statements for load balancing:**

**1 Record a multi-protocol script.**

Record a multi-protocol script for Oracle NCA and Web Protocols. Perform the desired actions and save the script.

**2 Define parameters for host and host arguments.**

Define two variables, serverHost and serverArgs, for parameterization:

```
web_set_max_html_param_len("512");
web_reg_save_param("serverHost", "NOTFOUND=ERROR",
      "LB=<PARAM name=\"serverHost\" value=\"","RB=\">", LAST);
web_reg_save_param("serverArgs", "NOTFOUND=ERROR",
      "LB=<PARAM name=\"serverArgs\" value=\"","RB=\">", LAST);
```

**3 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:\\appsnca…fndnam=apps ");
```

to:

```
nca_connect_server("< serverHost >", "9000"/*version=107*/, "<
serverArgs >");
```

The script should now look like this:

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

## 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.run-
forms\?ICX_TICKET=5843A55058947ED3&RESP_APP=AR&RESP_KE
Y=RECEIVABLES_MANAGER&SECGRP_KEY=STANDARD", LAST);
```

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

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

…

web_url("fnd_icx_launch.runforms",
"URL=http://ABC-123:8002/pls/VIS/fnd_icx_launch.run-
forms\?ICX_TICKET=<icx_ticket>&RESP_APP=AR&RESP_KEY=RECE
IVABLES_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" (RelFrameId=1)
vuser_init.c(8):
/servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdroot=my1san
w2n1.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("NCAJServSes-
sionId","LB=\r\n\r\n","RB=\r","ORD=1",LAST);

web_url("f60servlet",
     "URL= http://ABC-"123/servlet/oracle.forms.servlet.ListenerServ-
let?ifcmd=getinfo&" "ifhost=mercury&ifip=123.45.789.12", LAST);

web_url("oracle.forms.servlet.ListenerSer",
     "URL=http://ABC-123<NCAJServSessionId>?ifcmd=getinfo&"
"ifhost=mercury&ifip=123.45.789.12", LAST);
```

## Recording in Pragma Mode

The client side of the Oracle NCA Vuser can be configured to send an additional header to the server named **Pragma**. The header is a counter that behaves in the following way: the initial message of the NCA handshake has a value of 1.

The messages that follow the handshake are counted, beginning with 3. The counter is incremented by 1 for each message sent by the client.

If the message received from the server is the type plain\text and the body of the message begins with ifError:#/#00, the client sends a 0 byte message to the server and the Pragma value changes its sign to a minus. This sign changes back after the client succeeds in receiving the information from the server.

Recording of the Pragma header is only supported in the multi-protocol mode (Oracle NCA and Web). You can identify the Pragma mode within the script's **default.cfg** file. When operating in Pragma mode, the UseServletMode is set to 2.

```
[HttpConnectMode]
UseHttpConnectMode=1
RelativeURL=<NCAJServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see "Configuring Oracle NCA Client Emulation Run-Time Settings" on page 883.

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.ListenerServ-
let?JServSessionIdss2ser"
      "vlet=gk5q79uqy1 HTTP/1.1\r\n"
      "Pragma: 1\r\n"
      ...
send buf110
      "POST /ss2servlet/oracle.forms.servlet.ListenerServ-
let?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.ListenerServ-
let?JServSessionIdss2ser"
      "vlet=gk5q79uqy1 HTTP/1.1\r\n"
      "Pragma: -12\r\n"
      ...
```

# 59

# Developing SAPGUI Vuser Scripts

In the growing field of ERP (Enterprise Resource Planning), SAP provides solutions allowing companies to manage all of their business processes. Mercury provides tools for testing SAP solution modules on both functional and load testing levels. This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). For information on testing solutions for mySAP Workplace and Portal clients, see Chapter 60, "Developing SAP-Web Vuser Scripts." This chapter describes:

- About Developing SAPGUI Vuser Scripts
- Checking your Environment for SAPGUI Vusers
- Creating a SAPGUI Vuser Script
- Recording a SAPGUI Vuser Script
- Setting the SAPGUI Recording Options
- Inserting Steps Interactively into a SAPGUI Script
- Understanding a SAPGUI Vuser Script
- Enhancing a SAPGUI Vuser Script
- Replaying SAPGUI Optional Windows
- Setting SAPGUI Run-Time Settings
- SAPGUI Functions
- Tips for SAPGUI Vuser Scripts
- Troubleshooting SAPGUI Vuser Scripts
- Additional Resources

***The following information only applies to the SAPGUI and the SAPGUI/SAP-Web dual protocols.***

# About Developing SAPGUI Vuser Scripts

This chapter discusses the solution for testing the SAPGUI for Windows client (SAPGUI Vuser). To test the SAPGUI user operating only on the client, use the SAPGUI Vuser type. To test a SAPGUI user that also uses a Web browser, use the SAPGUI/SAP-Web dual protocol.

Before recording a session, verify that your modules and client interfaces are supported by VuGen. The following sections describe the SAP client modules for SAP Business applications.

- SAP Web Client or mySAP.com: Use the SAP-Web Vuser type.
- SAPGUI for Windows - a Windows-based client, emulated by the SAPGUI Vuser. This also supports APO module recording (requires patch level 24 for APO 3.0).
- SAPGUI for Windows and a web browser: Use the SAPGUI/SAP-Web dual protocol.
- SAPGUI for Java: This client is not supported

**Version 6.20 and later**:

- **For Functional Testing:** Use the QuickTest Professional Add-in for mySAP.com client.
- **For Load Testing:** Use the SAPGUI or SAPGUI/SAP-Web dual protocol to create a script in VuGen and run a scenario or session step in the Controller or Console.

You use VuGen's recorder to record typical business processes. VuGen records SAPGUI for Windows client activity during SAP business processes, and generates a Vuser script. When you perform actions within the SAPGUI for Windows client, VuGen generates functions that describe this activity. Each function begins with a **sapgui** prefix.

During replay, these functions emulate user activity on SAPGUI objects.

For example, **sapgui_select_radio_button** selects the radio button **Blue**.

```
sapgui_select_radio_button("Blue",
        "usr/radRB7",
        BEGIN_OPTIONAL,
           "AdditionalInfo=sapgui1027",
        END_OPTIONAL);
```

# Checking your Environment for SAPGUI Vusers

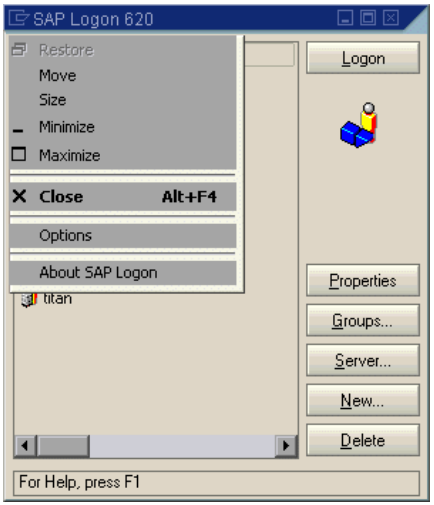
The basic steps in checking and setting up your system for the recording of SAPGUI Vusers, are Checking the Patch Level and Enabling Scripting. Once your environment is configured properly, you can record a typical SAP session and replay it in VuGen.

## Checking the Patch Level

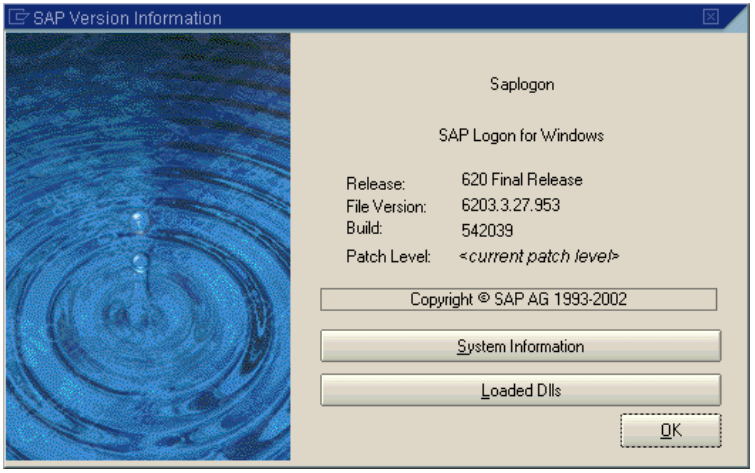
You can check the patch level of your SAPGUI for Windows client from the About box. The lowest patch level supported is 32.

**To check the patch level:**

**1** Invoke the SAPGUI logon window. Click the top left corner of the SAP Logon dialog box and choose **About SAP Logon** from the menu.




**2** The SAP version information dialog box opens. Verify that the Patch Level entry is 32 or higher.



## Enabling Scripting

Mercury Interactive's support for the SAPGUI for Windows client, is based on SAP's Scripting API. This API allows Vusers to interact with the SAPGUI client, receive notifications, and perform operations.

The Scripting API is only available in recent versions of the SAP Kernel. In kernel versions that support scripting, the option is disabled by default. In order to use Mercury Interactive's tools, first ensure that the SAP servers support the Scripting API, and enable the API on both the server and clients. For more information and to download patches, refer to the SAP OSS note #480149.

VuGen provides a utility that checks if your system supports scripting. The utility, **VerifyScript.exe** is located on the CD in the **Patches and Tools** directory. For more information, refer to the Readme file provided with this utility.

The following sections present the steps that are required to enable scripting.

- Checking the Configuration
- Enabling Scripting on the SAP Application Server
- Enabling Scripting on SAPGUI 6.20 Client

### Checking the Configuration

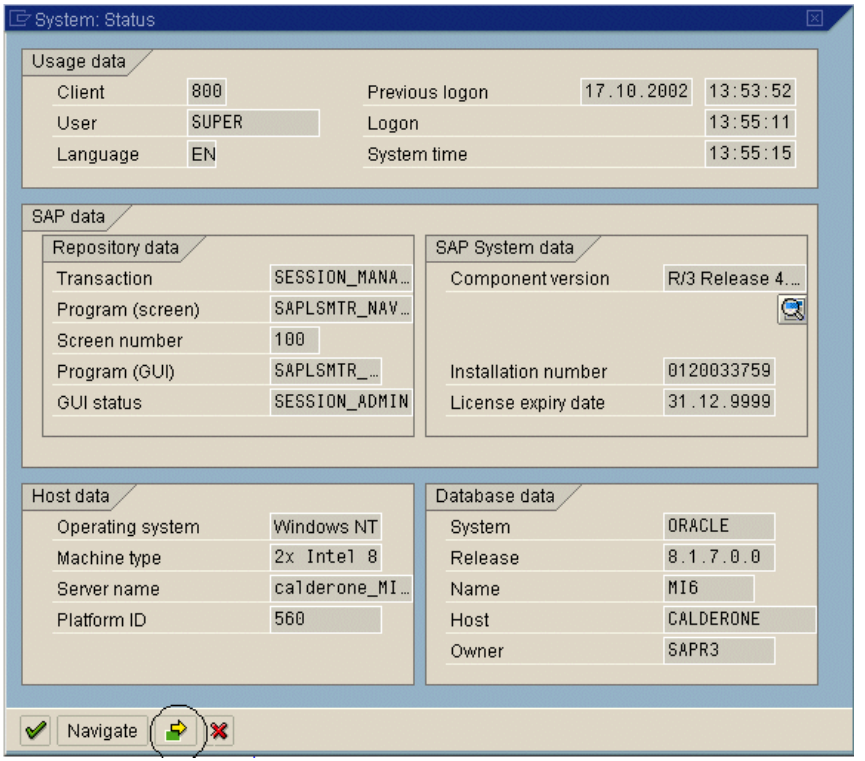
The first step in enabling scripting is ensuring that the right kernel version is installed, and updating it if required.

Check the table below, for the minimum kernel patch level required for your version of the SAP Application Server. If required, download and install the latest patch.

| Software Component | Release | Package Name | Kernel Patch Level |
|---|---|---|---|
| SAP_APPL | 31I | SAPKH31I96 | Kernel 3.1I level 650 |
| SAP_APPL | 40B | SAPKH40B71 | Kernel 4.0B level 903 |
| SAP_APPL | 45B | SAPKH45B49 | Kernel 4.5B level 753 |
| SAP_BASIS | 46B | SAPKB46B37 | Kernel 4.6D level 948 |
| SAP_BASIS | 46C | SAPKB46C29 | Kernel 4.6D level 948 |
| SAP_BASIS | 46D | SAPKB46D17 | Kernel 4.6D level 948 |
| SAP_BASIS | 610 | SAPKB61012 | Kernel 6.10 level 360 |

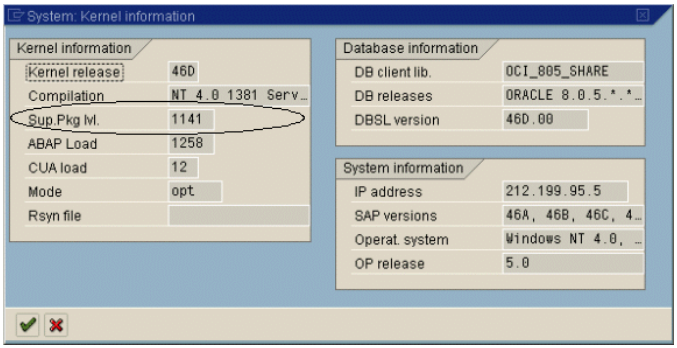
**To check the kernel patch level:**

1. Log in to the SAP system
2. Select **System** > **Status**
3. Click the **Other kernel information button** (with the yellow arrow).



**4** In the **Kernel Information** section, check the value of the **Sup. Pkg. lvl.**

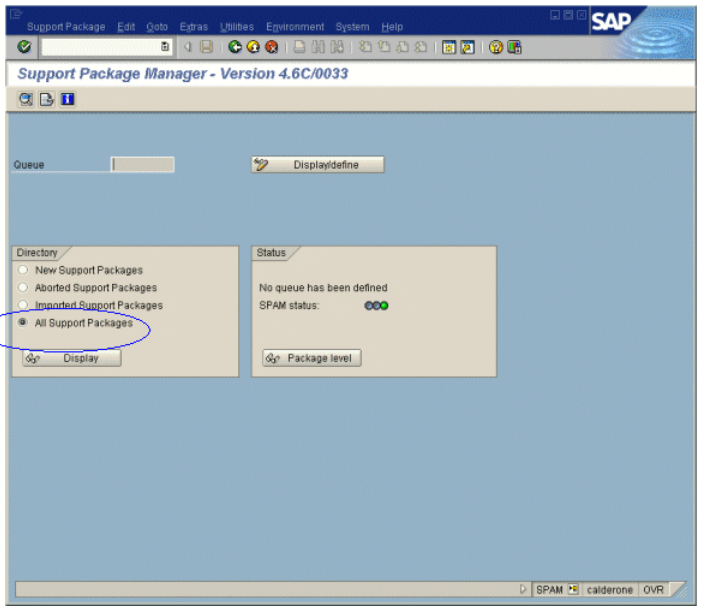
If the level is lower than 948, you must download the latest kernel version and upgrade your existing one. Refer to the SAP OSS note #480149 for detailed instructions on how to perform this upgrade.



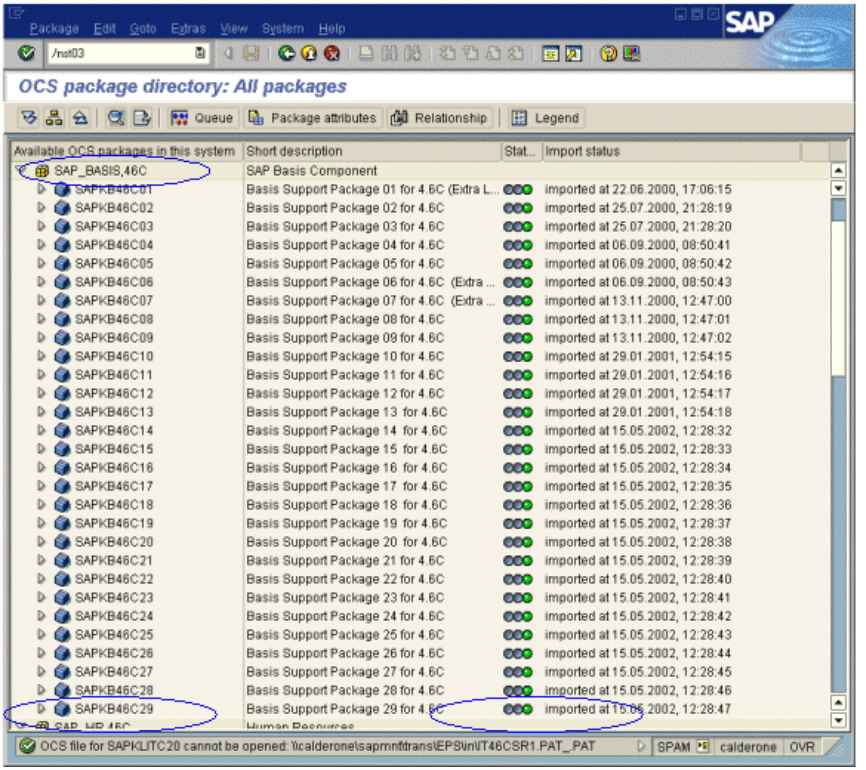

**To check the R/3 support packages:**

**1** Log on to the SAP system and run the SPAM transaction.

**2** In the **Directory** section, select **All Support Packages**, and click the **Display** button.



**3** Verify that SAPKB46C29 is installed for SAP_BASIS, 4.6C. If it is installed, a green circle appears in the Status column.



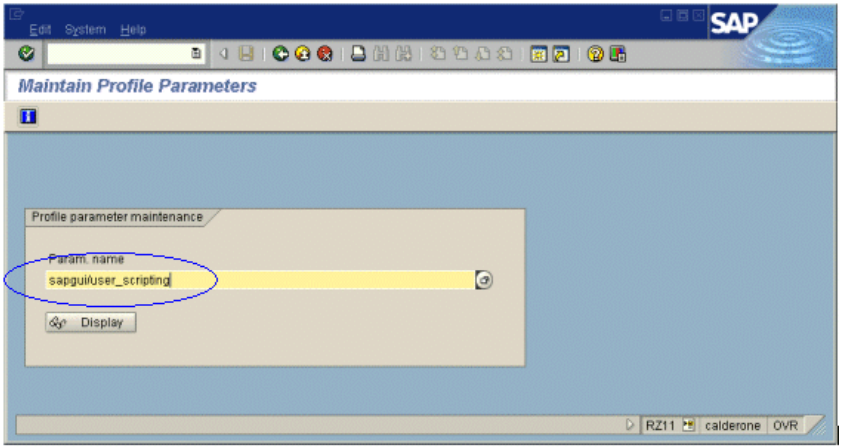

If you do not have the OCS package installed, download it the from the www.sap.com Web site and install it. For more information, refer to the SAP OSS note #480149.

### Enabling Scripting on the SAP Application Server

A user with administrative permissions enables scripting by setting the **sapgui/user_scripting** profile parameter to **TRUE** on the application server. To enable scripting for all users, set this parameter on all application servers. To enable scripting for a specific group of users, only set the parameter on application servers with the desired access restrictions.

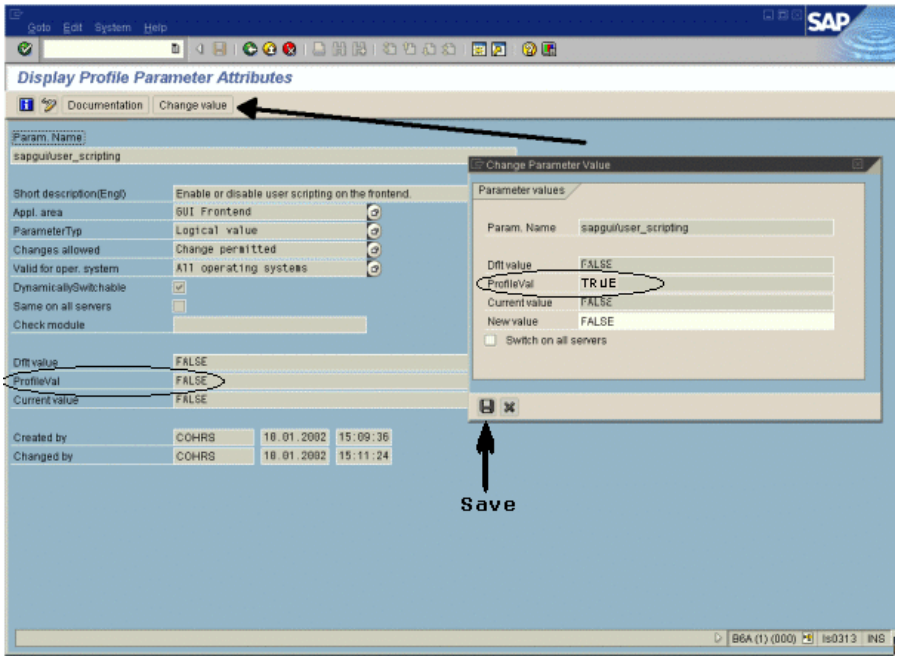
**To change the profile parameter:**

1 Open transaction **rz11.** Specify the parameter name **sapgui/user_scripting** and click **Display**. The Display Profile Parameter Attributes window opens.




If **Parameter name is unknown** appears in the status bar, this indicates that you are missing the current Support Package. Import the Support Package that corresponds to the SAP BASIS and kernel versions of the application server, as described in “Checking the Configuration” on page 900.

**2** If **Profile Val** is FALSE, you need to modify its value. Click the **Change value** button in the toolbar. The Change Parameter Value window opens. Enter TRUE in the **ProfileVal** box and click the **Save** button.




When you save the change, the window closes and **ProfileVal** is set to TRUE.

**3** Restart the application server, since this change only takes effect when you log onto the system.

If the updated **ProfileVal** did not change, even after restarting the server, then the kernel of the application server is outdated. Import the required kernel patch, as specified in the section "Checking the Configuration" on page 900.

Note that the Profile Value may be dynamically activated in the following kernel versions, using transaction rz11, without having to restart the application server.

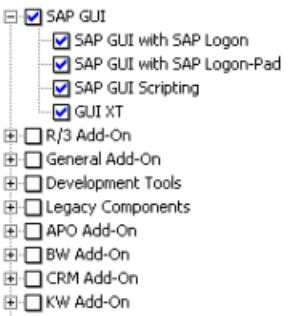
| Release | Kernel Version | Patch Level |
|---|---|---|
| 4.6B, 4.6C, 4.6D | 4.6D | 972 |
| 6.10 | 6.10 | 391 |
| 6.20 | all versions | all levels |

### Enabling Scripting on SAPGUI 6.20 Client

To allow VuGen to run scripts, you must also enable scripting on the SAPGUI client. You should also configure the client not to display certain messages, such as when a connection is established, or when a script is attached to the GUI process.

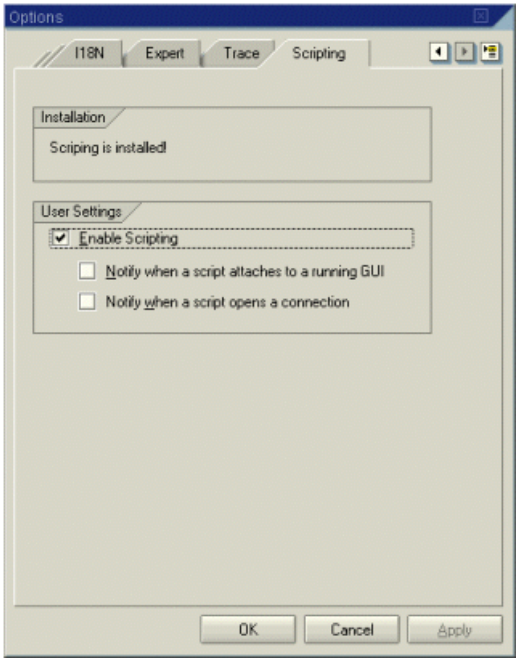
**To configure the SAPGUI client to work with VuGen:**

➤ **During installation:** While installing the SAPGUI client, enable the **SAP GUI Scripting** option.




➤ **After installation:** Suppress warning messages. Open the Options dialog box in the SAPGUI client. Select the **Scripting** tab and clear the following options:

1. **Notify when a script attaches to a running GUI**
2. **Notify when a script opens a connection**




You can also prevent these messages from popping up by setting the values **WarnOnAttach** and **WarnOnConnection** in the following registry key to 0:

HKCU\SOFTWARE\SAP\SAPGUI Front\SAP Frontend Server\Security.

# Creating a SAPGUI Vuser Script

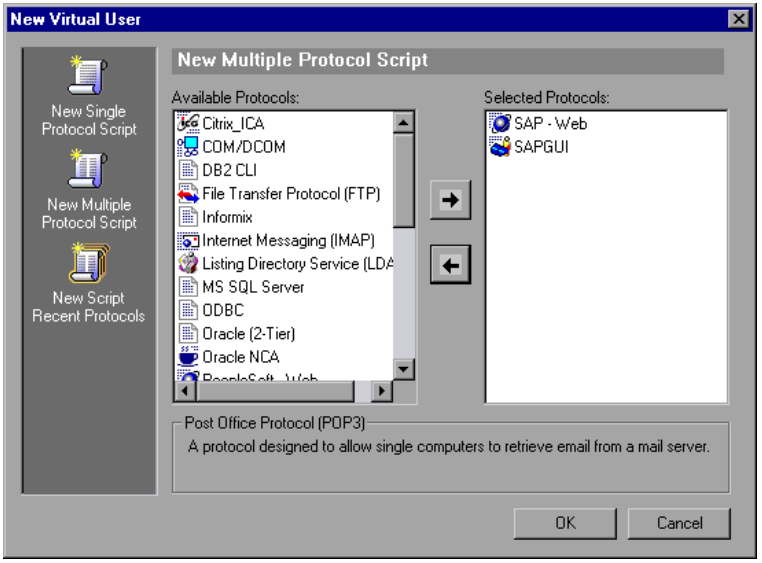
The first step in creating a SAPGUI Vuser script is choosing the Vuser and script type. The SAP Vuser type, **SAPGUI** is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script.

**To create a SAPGUI Vuser script:**

1 Invoke VuGen and choose **File** > **New**.

2 To record a standard SAPGUI client session (with no browser controls), create a single-protocol Vuser script using the **SAPGUI** type Vuser.

3 To record a SAPGUI session that uses browser controls, create a multi-protocol Vuser script. Specify both the **SAPGUI** and **SAP-Web** Vuser types. This allows VuGen to record Web-specific functions when encountering the browser controls.




4 Click **OK** to open the Vuser script.

# Recording a SAPGUI Vuser Script

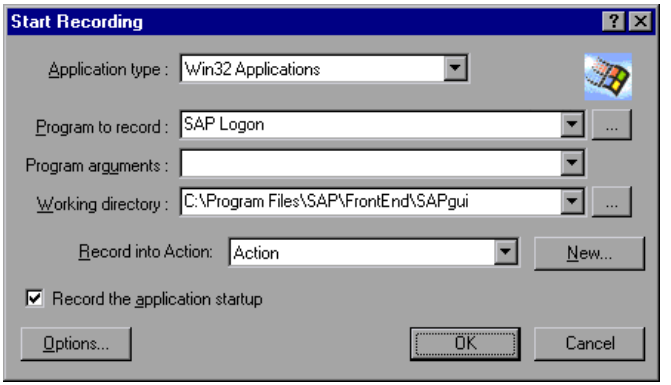
After creating an empty script, you set the recording options and then record your SAPGUI session. VuGen generates a script corresponding to your actions within the client.

**To begin recording a SAPGUI script:**

**1** If the Start Recording dialog box was not opened, click the **Start Recording** button. The Start Recording dialog box opens.



**2** VuGen detects and fills in the relevant information:




**Program to record:** VuGen locates the **saplogon.exe** file in the SAP client installation.

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

**Record into Action:** Select the section into which you want to record. Initially, the available sections are **vuser_init**, **Action1**, and **vuser_end**.

**3** Click **OK** and begin recording.

### Recording at the Cursor

VuGen also allows you to record actions into an existing script. You may choose to record into an existing script for several reasons:

- You made a mistake in the actions that you performed during recording.
- Your actions were correct, but you need to add additional information such as the handling of popup windows. For example the SAP server may issue an inventory warning, which did not apply during the recording session.

This feature, called Recording at the Cursor, lets you insert new actions or replace existing actions. When you begin Recording at the Cursor, VuGen prompts you with two options:

**Insert steps into action:** Inserts the newly recorded steps at the cursor without overwriting any existing steps. The new segment is enclosed with comments indicating the beginning and end of the added section. This option is ideal for handling occasional popup windows that were not present during the recording

```
// Recording at the cursor - Begin
    sapgui_select_active_connection("con[0]");
    sapgui_select_active_session("ses[0]");
    sapgui_select_active_window("wnd[0]");
//Recording at the cursor - End
```

**Overwrite the rest of the script:** Replaces all steps from the point of the cursor onward. This option overwrites the remainder of the current Action and deletes all other Actions. It does not affect the **vuser_init** or **vuser_end** sections. This option is ideal for when you make a mistake in the recording.

After you choose one of the Recording at the Cursor options, VuGen replays the script from the beginning until the cursor's entry point. Then it opens the Recording floating toolbar and begins recording.

---

**Note:** If you use the Recording at the Cursor feature, the **Regenerate Script** tool becomes disabled.

---

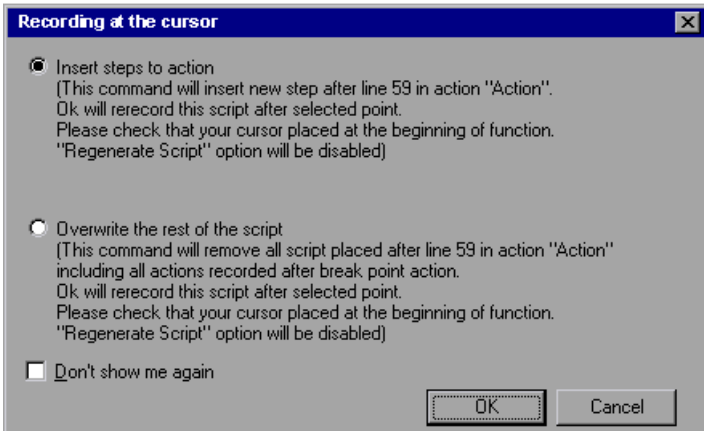
**To Record at the Cursor:**

Note that you can choose a default and then instruct VuGen to hide this dialog box for future Recording at the Cursor.

**1** Click the **Recording at the Cursor** button.




VuGen prompts you to make a selection.




**2** Select **Insert steps into action** or **Overwrite the rest of the script**. Click **OK**. VuGen replays the script until the point of the cursor.

**3** Wait for the Recording floating toolbar to open. Then begin performing actions in the SAPGUI client, switching between sections and actions as required.






**4** Click the **Stop** button to end the recording session.

# Setting the SAPGUI Recording Options

You use the recording options to set your SAP-related preferences for the recording session. To open the Recording Options dialog box, choose **Tools** > **Recording Options** or click **Options** in the Start Recording dialog box. The keyboard shortcut is CTRL+F7.

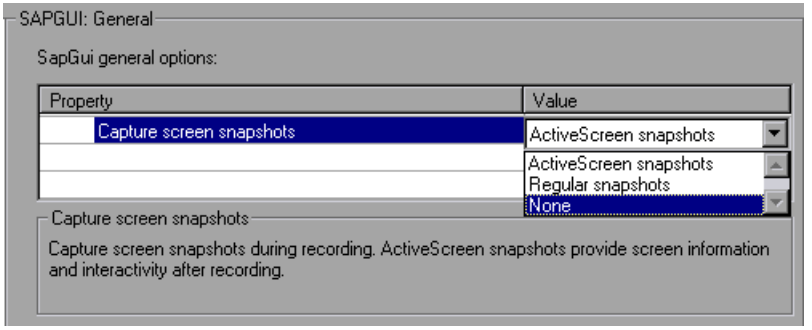
You can set recording options in the following areas:

- SAPGUI General Recording Options
- SAPGUI Code Generation Recording Options
- SAPGUI Auto Logon Recording Options

If you are recording a multi-protocol Vuser script with a SAP-Web Vuser type, see Chapter 40, "Setting Recording Options for Internet Protocols" for additional recording options.

## SAPGUI General Recording Options

You use these recording options to set your general preferences during the recording session.



**To set the General recording options:**

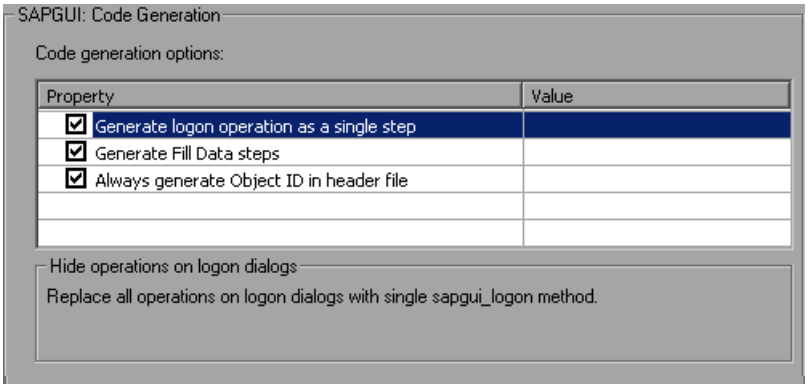
1 Open the Recording Options dialog box and select the **SAPGUI:General** node.

2 For the **Capture screen snapshots** option, indicate how to save the snapshots of the SAPGUI screens as they appear during recording. Select an item from the list: **ActiveScreen snapshots**, **Regular snapshots**, or **None**.

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

## SAPGUI Code Generation Recording Options

You use these recording options to set the code generation preferences.




**To set the Code Generation recording options:**

1 Open the Recording Options dialog box and select the **SAPGUI:Code Generation** node.

2 Select **Generate logon operation as a single step** to instruct VuGen to generate a single **sapgui_logon** method for all of the logon operations. This helps simplify the code. If you encounter login problems, disable this option.

3 To generate Fill Data steps for table and grid controls—instead of separate steps for each cell, select **Generate Fill Data Steps**.

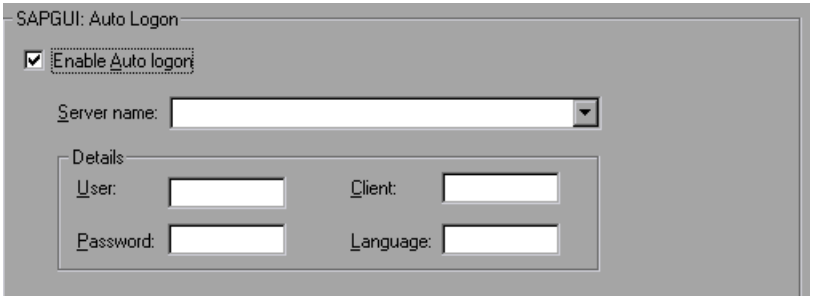
4 To create a more compact and cleaner script, select **Always generate Object IDs in header file** which places the Object IDs in a separate header file instead of in the script. When you disable this option, VuGen generates the

IDs according to the specified string length in the general script setting. Note that disabling this option only increases readability—there is no difference in overhead.

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

### SAPGUI Auto Logon Recording Options

You set these recording options to log on automatically when you begin recording. The logon functions are placed in the vuser_init section of the script. The server name list contains all of the servers on the SAP Logon description list




**To enable and set the Auto Logon recording options:**

**1** Open the Recording Options dialog box and select the **SAPGUI:Auto Logon** node.

**2** Select **Enable Auto logon**.

**3** Enter the Login information:

- the SAP **Server name**
- the **User** name for the SAP server
- the **Password** for the SAP server
- the **Client** name by which the SAP server identifies the client
- the interface **Language**

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

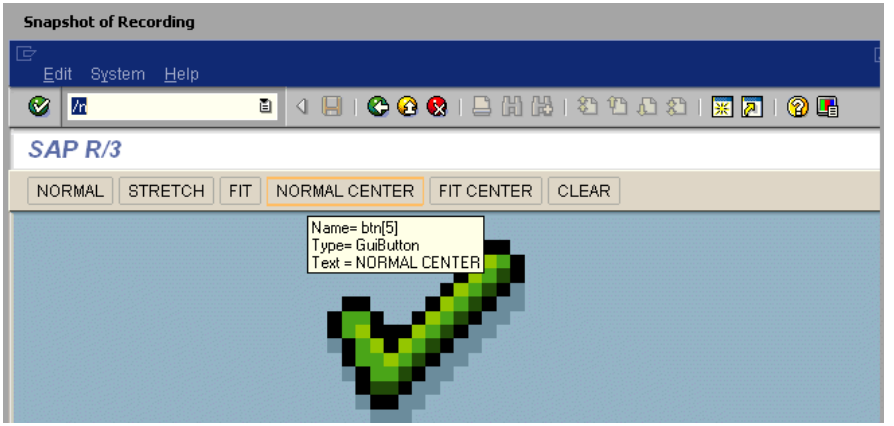
# Inserting Steps Interactively into a SAPGUI Script

After recording, you can manually add steps to the script in either Script view and Tree View. For information about adding steps from the various views, see “Viewing and Modifying Vuser Scripts” on page 17.

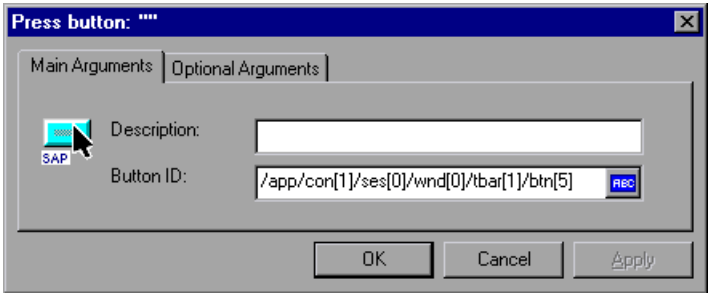
In addition to manually adding new functions, you can add new steps interactively for SAPGUI Vusers, directly from the snapshot. Using the right-click menu, you can add object-related steps.

When adding a step from within a snapshot, VuGen uses the Active Screen capability and determines the ID of each object in the SAPGUI client window (unless you disabled Active Screen snapshots in the SAPGUI General Recording Options).

To determine which objects were recognized by VuGen, you move the mouse over the snapshot. VuGen draws a box around the objects as you pass over them and displays a tool tip with the object’s Control ID. In the following example, the selected active object is the NORMAL CENTER button.



When you add a step while holding the mouse over a recognized object, VuGen automatically inserts the Control ID of that object into the relevant field of the Properties dialog box. For example, if you add a **Press Button** step, for the NORMAL CENTER button as shown above, the Properties box displays the following ID:



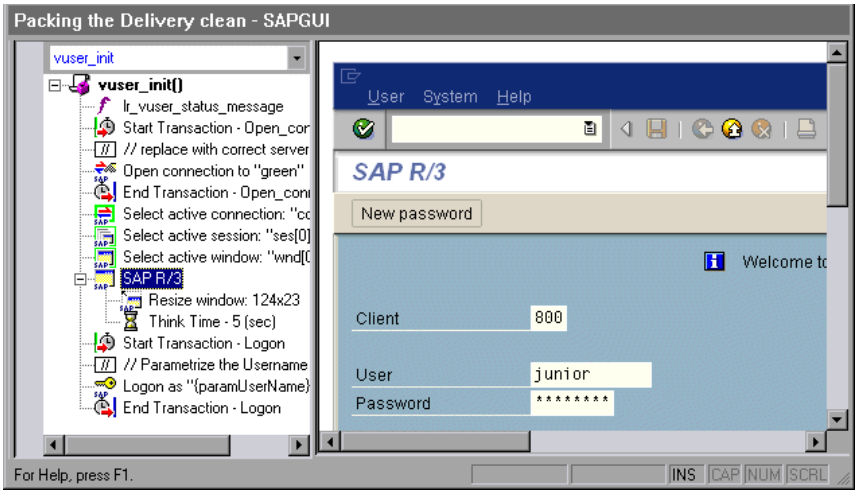

**To insert a step interactively for a specific object:**

1. Click within the Snapshot window.
2. Move the mouse over the object for which you want to add a function. Make sure that VuGen recognizes the object and encloses it with a box.
3. Select **Insert New Step** from the right-click menu. The Insert Step box opens.
4. Choose a step from the menu. The step's Properties dialog box opens, with the Control ID of the object when relevant.
5. Enter a name for the object in the **Description** box. Click **OK**. VuGen inserts the new step after the selected step.
6. To get the Control ID of the object for the purpose of pasting it into a specific location, select Copy **Control ID** from the right-click menu. VuGen places it on the clipboard. You can past it into a Properties box or directly into the code from the Script view.

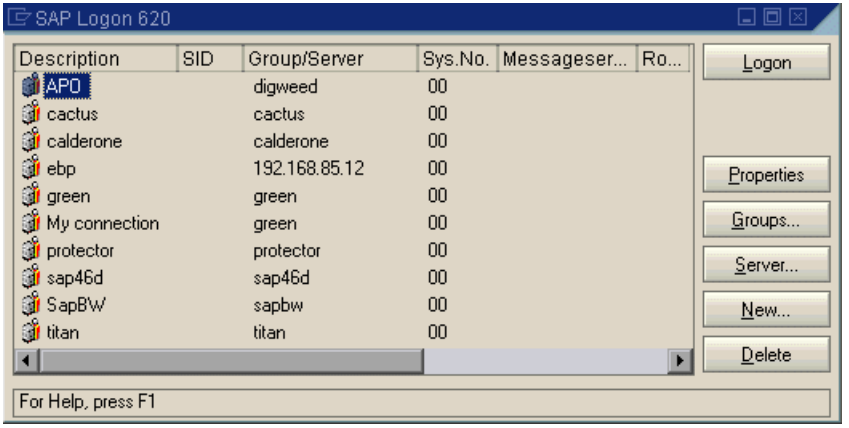
# Understanding a SAPGUI Vuser Script

The SAPGUI Vuser script typically contains several SAP transactions which make up a business process. A business process consists of functions that emulate user actions. Open the tree view to see each user action as a Vuser script step.

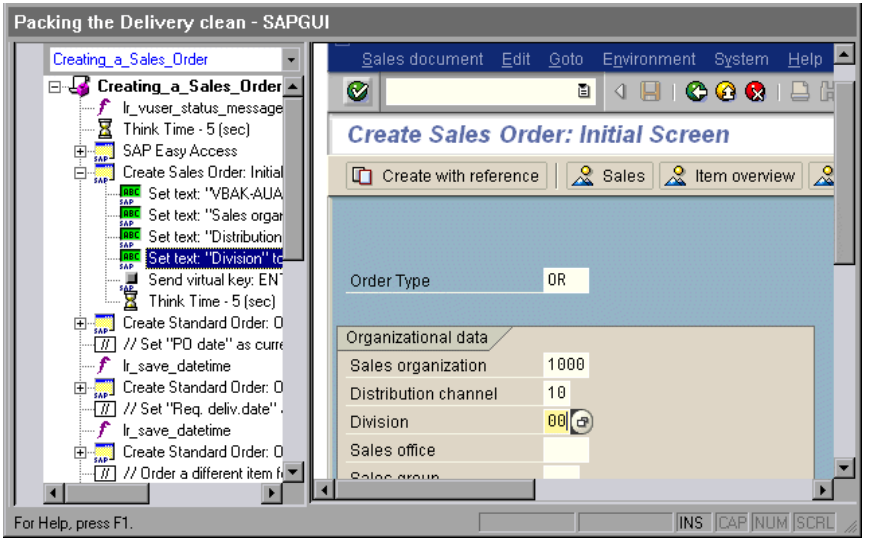
The following example shows a typical recording of a SAPGUI client. The first section, **vuser_init**, contains the opening of a connection and a logon.



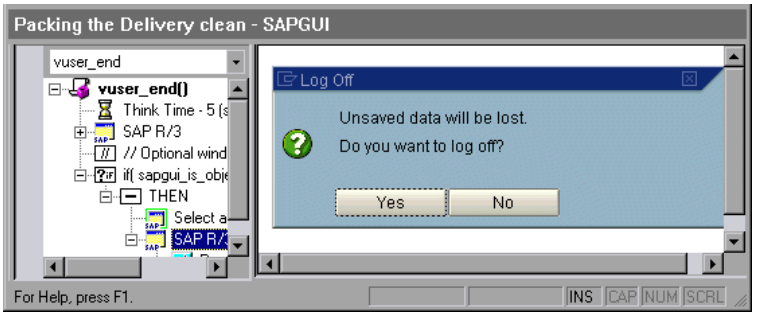
Note that the Open Connection step uses one of the connection names in the SAP Logon **Descriptions** list. If the specified connection name is not in the list, the Vuser looks for a server with that name.




In the following section, the functions emulate typical user operations such as menu selection and the setting of a check box.



The final section, **vuser_end**, illustrates the logoff procedure.




When recording a multi- protocol script for both SAPGUI and Web, VuGen generates steps for both protocols. In the Script view, you can view both **sapgui** and **web** functions.

The following example illustrates a multi-protocol recording in which the SAPGUI client opens a Web control. Note the switch from **sapgui** to **web** functions.

```
sapgui_tree_double_click_item("Use as general WWW browser, REPTI-
TLE",
      "shellcont/shell",
      "000732",
      "REPTITLE",
      BEGIN_OPTIONAL,
           "AdditionalInfo=sapgui1020",
      END_OPTIONAL);

...
sapgui_set_text("",
      "http:\\\\yahoo.com",
      "usr/txtEDURL",
      BEGIN_OPTIONAL,
           "AdditionalInfo=sapgui1021",
      END_OPTIONAL);

...
web_add_cookie("B=7pt5cisv1p3m2&b=2; DOMAIN=www.yahoo.com");

web_url("yahoo.com",
           "URL=http://yahoo.com/",
           "Resource=0",
           "RecContentType=text/html",
           "Referer=",
           "Snapshot=t1.inf",
           "Mode=HTML",
           EXTRARES,
"URL=http://srd.yahoo.com/hpt1/ni=17/ct=lan/sss=1043752588/t1=10437
52575385/d1=1251/d2=1312/d3=1642/d4=4757/0.4097009487287739/*1"
, "Referer=http://www.yahoo.com/", ENDITEM,
           LAST);
```

# Enhancing a SAPGUI Vuser Script

After you examine the recorded Vuser script, you enhance it in the following ways:

- **Transactions**: Inserting transactions, rendezvous points, and control-flow structures into the script. For details, see Chapter 7, "Enhancing Vuser Scripts."
- **Verification:** Insert SAPGUI verification functions to verify the current state of SAPGUI objects. For details, see Adding Verification Functions.
- **Retrieve information:** Insert SAPGUI functions to verify the current value of SAPGUI objects. You use the **sapgui_get_xxx** functions to retrieve information. For more information, see "Retrieving Information" on page 923.

**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 8, "Working with VuGen Parameters."

## Adding Verification Functions

When working with optional or dynamic windows or frames, you can use verification functions to determine if the window or object is available. An optional window is a window that does not consistently open during the SAP session. This function allow the Vuser script to continue running even if an optional window opens or an exception occurs.

The first example checks if a window is available. If the window is available, the Vuser closes it before continuing.

```
if (!sapgui_is_object_available("wnd[1]"))
        sapgui_call_method("{ButtonID}",
                "press",
                LAST,
                AdditionalInfo=info1011");
sapgui_press_button(.....)
```

The next example illustrates a dynamic object in the ME51N transaction. The Document overview frame is optional, and can be opened/closed by the **Document overview on/off** button.

The code checks the text on the Document overview button. If the text on the button shows **Document overview on**, we click the button to close the Document overview frame.

```
    if(sapgui_is_object_available("tbar[1]/btn[9]"))
    {
        sapgui_get_text("Document overview on/off button",
                "tbar[1]/btn[9]",
                "paramButtonText",
                LAST);

        if(0 == strcmp("Document overview off", lr_eval_string("{param-
ButtonText}")))
                sapgui_press_button("Document overview off",
                    "tbar[1]/btn[9]",
                    BEGIN_OPTIONAL,
                        "AdditionalInfo=sapgui1013",
                    END_OPTIONAL);
    }
```

## Retrieving Information

When working with SAGUI Vusers, you can retrieve the current value of a SAPGUI object using the **sapgui_get_<xxx>** functions. You can use this value as input for another business process, or display it in the output log.

### Retrieving Status Bar Information

The following example illustrates how to save part of a status bar message in order to retrieve the order number.

**To retrieve the order number from the status bar:**

1. Navigate to the point where you want to check the status bar text, and select **Insert > New Step.** Choose the **sapgui_status_bar_get_type** function. This verifies that the Vuser can successfully retrieve text from the status bar.
2. Insert an **if** statement that checks if the previous statement succeeded. If so, save the value of the argument using **sapgui_status_bar_get_param**.

   This **sapgui_status_bar_get_param** function saves the order number into a user-defined parameter. In this case, the order number is the second index of the status bar string.

```
sapgui_press_button("Save (Ctrl+S)",
      "tbar[0]/btn[11]",
      BEGIN_OPTIONAL,
           "AdditionalInfo=sapgui1038",
      END_OPTIONAL);

sapgui_status_bar_get_type("Status");
if(0==strcmp(lr_eval_string("{Status}"),"Success"))
  sapgui_status_bar_get_param("2", " Order_Number ");
```

During test execution, the Execution log indicates the value and parameter name:

```
Action.c(240): Pressed button " Save (Ctrl+S)"
Action.c(248): The type of the status bar is "Success"
Action.c(251): The value of parameter 2 in the status bar is "33232"
```

### Saving Date Information

When creating scripts that use dates, your script may not run properly. For example, if you record the script on June 2, and replay it on June 3, the date fields will be incorrect. Therefore, you need to save the date to a parameter during text execution, and use the stored value as input for other date fields. To save the current date or time during script execution, use the **lr_save_datetime** function. Insert this function before the function requiring the date information. Note that the format of the date is specific to your locale. Use the relevant format within the **lr_save_datetime** function. For example, for month.day.year, specify "%m.%d.%Y".

In the following example, **lr_save_datetime** saves the current date. The **sapgui_set_text** function uses this value to set the delivery date for two days later.

```
lr_save_datetime("%d.%m.%Y", DATE_NOW + (2 * ONE_DAY),
    "paramDateTodayPlus2");

sapgui_set_text("Req. deliv.date",
            "{paramDateTodayPlus2}",
            "usr/ctxtRV45A-KETDAT",
            BEGIN_OPTIONAL,
                    "AdditionalInfo=sapgui1025",
            END_OPTIONAL);
```

## Replaying SAPGUI Optional Windows

When working with SAPGUI Vuser Scripts, you may encounter optional windows in the SAPGUI client—windows that were present during recording, but do not exist during replay. If you try to replay your recorded script as is, it will fail when it attempts to find the missing windows.

VuGen's optional window mechanism performs the actions on a window only after verifying that it exists. The Vuser checks if the window indicated in the **Select active window** step exists. If the window is found during replay, it performs the actions as they were recorded in the script. If it does not exist, the Vuser ignores all window actions until the next **Select active window** step. Note that only SAPGUI steps (beginning with a **sapgui** prefix) are ignored.

To use this feature, in Tree view select the appropriate Select Active Window step and choose **Run steps for window only if it exists** from the right-click menu.

To disable this feature and attempt to run these steps at all times, regardless of whether the Vuser finds the window or not, choose **Always run steps for this window** from the right-click menu.

## Setting SAPGUI Run-Time Settings

After creating and enhancing your SAPGUI Vuser script, you configure its run-time settings and run it from VuGen to check its functionality. Run-Time settings let you control the Vuser behavior during replay. You configure these settings before running the Vuser script. You can set both general and SAPGUI-specific run-time settings.

The general settings include the run logic, pacing, logging, think time, and performance preferences. For information about the general run-time settings, see Chapter 12, "Configuring Run-Time Settings." For SAPGUI-specific settings, see the following sections.

Once you configure the Run-Time settings, you save the Vuser script and run it from VuGen to verify that it runs correctly. For details about running the Vuser script as a standalone test, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

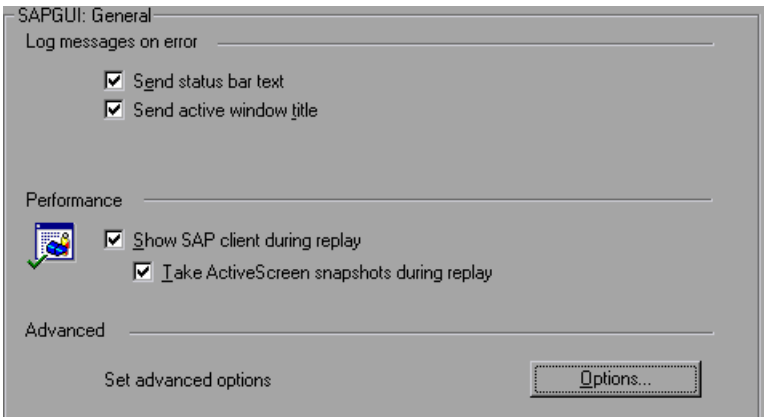
After verifying that your Vuser script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

You can configure the SAPGUI specific Run-Time settings in the following areas:

- SAPGUI General Run-Time Settings
- SAPGUI Advanced Run-Time Settings

### SAPGUI General Run-Time Settings

General run-time settings let you set the general settings for a SAPGUI Vuser script. VuGen uses these settings when running the script.




The Log run-time settings specify the information a Vuser sends to the Execution log whenever an error occurs.

**Send status bar text:** Send the text from the status bar to the log file.

**Send active window title:** Send the active window title text to the log file.

The Performance run-time settings allow you to indicate whether or not to display the SAP client during replay.

**Show SAP Client during replay:** Shows an animation of the actions in the SAP client during replay. The benefit of displaying the user interface (UI) is that you can see how the forms are filled out and closely follow the actions of the Vuser. This option, however, requires additional resources and may affect the performance of your load test.

**Take ActiveScreen snapshots during replay:** Captures replay snapshots with the Control ID information for all active objects. ActiveScreen snapshots differ from regular ones, in that they allow you to see which objects were recognized by VuGen in the SAPGUI client. As you move your mouse across the snapshot, VuGen highlights the detected objects. You can then add new steps to the script directly from within the snapshot. It also allows you to add steps interactively from within the snapshot for a specific object. For more information, see "Inserting Steps Interactively into a SAPGUI Script" on page 915.

Advanced options let you set a timeout for the **SAPfewgsvr.exe** process, save a snapshot on error, and configure VuGen to use SAPlogon during replay. For more information, see "SAPGUI Advanced Run-Time Settings" on page 928.

**To set the SAPGUI Run-Time Settings:**

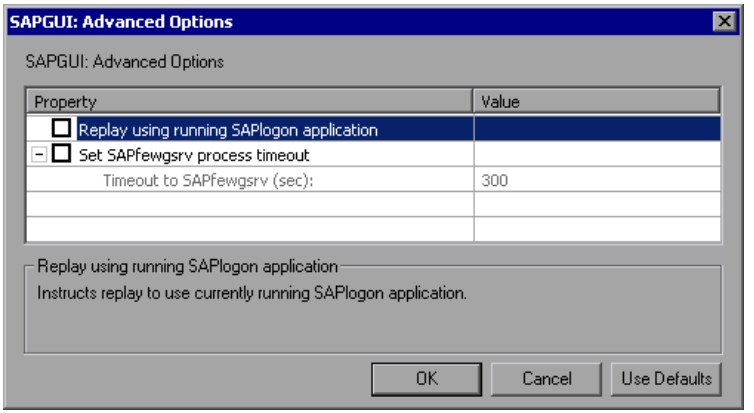


1. Open the Run-Time settings dialog box. Click the **Run-Time Settings** button on the VuGen toolbar, or choose **Vuser** > **Run-Time Settings**.
2. Select the **SAPGUI:General** node.
3. In the **Log messages on error** section, select one or more message sources: **Send status bar text** or **Send active window title**.
4. In the **Performance** section, select the **Show SAP client during replay** check box to show the SAPGUI user interface during replay.
5. Click **Options** to set a timeout for the **SAPfewgsvr.exe** process.

## SAPGUI Advanced Run-Time Settings

Each Vuser invokes a separate **SAPfewgsvr.exe** process during test execution. In some instances, the process stays active even after the replay session has ended. You can check the Windows Task Manager to see if the process is still active.

The Advanced SAPGUI settings let you set a timeout for this application. When the timeout is reached, VuGen closes any **SAPfewgsvr** processes not previously terminated.




**Replay using running SAPlogon application**: Instructs the Vusers to use the SAPlogon application that is currently running for replay.

**Set SAPfewgsvr application timeout**: Allows you to modify the **SAPfewgsvr.exe** process timeout.

➤ **Timeout to SAPfewgsvr:** The **SAPfewgsvr.exe** process timeout in seconds. The default is 300 seconds.

# SAPGUI Functions

During a SAPGUI recording session, VuGen generates functions that emulate user interaction with the SAPGUI client. When you record the SAPGUI for Windows client, VuGen generates functions with a **sapgui** prefix. This section lists all of the **sapgui** functions.

When you record a SAP session using a Web interface such as SAP Workplace or Portal, or if the SAPGUI client opens a Web control, VuGen generates functions with a **web** prefix.

For more information about the **sapgui** and **web** functions, use the **Show Function Syntax** feature from the Edit menu, or refer to the *Online Function Reference* (**Help** > **Function Reference**).

While most of the functions are recorded, you can manually insert any function into your script. The functions that are not recorded are the data retrieval functions beginning with **sapgui_get**, and those used for verification, beginning with **sapgui_is**.

There are several categories of **sapgui** functions: Connection and Session Functions, Method and Property Functions, Verification and Data Retrieval Functions, and Object functions. Object functions are those which perform an action within a SAPGUI object such as Calendar Functions, Grid Functions, APO Grid Functions, Status Bar Functions, Table Functions, Tree Functions, Window Functions, and General Object Functions.

### Connection and Session Functions

| | |
|---|---|
| **sapgui_create_session** | Creates a new SAPGUI session. |
| **sapgui_logon** | Logs in to a SAP server. |
| **sapgui_open_connection** | Opens a connection to a SAP server. |
| **sapgui_open_connection_ex** | Opens a connection to the SAP server specified by a connection string. |

| | |
|---|---|
| **sapgui_select_active_connection** | Sets the specified connection as the active connection. |
| **sapgui_select_active_session** | Sets the active SAPGUI session. |

### Method and Property Functions

| | |
|---|---|
| **sapgui_get_property_of_active_object** | Retrieves a property of the active object. |
| **sapgui_active_object_from_parent _method** | Selects an object within a parent object by calling the parent's method. |
| **sapgui_active_object_from_parent _property** | Selects an object that is a property of a parent object. |
| **sapgui_call_method** | Invokes a method of a SAPGUI object. |
| **sapgui_call_method_of_active_object** | Invokes a method of the active object. |
| **sapgui_get_property** | Gets the property of a SAPGUI object. |
| **sapgui_set_collection_property** | Sets the property of a object of type SAP GuiCollection. |
| **sapgui_set_property** | Sets the property of a SAPGUI object. |
| **sapgui_set_property_of_active_object** | Sets a property of the active object. |

## APO Grid Functions

| | |
|---|---|
| **sapgui_apogrid_clear_selection** | Deselects all selected cells. |
| **sapgui_apogrid_deselect_cell** | Deselects a specific cell. |
| **sapgui_apogrid_deselect_column** | Deselects a specific column. |
| **sapgui_apogrid_deselect_row** | Deselects a specific row. |
| **sapgui_apogrid_double_click** | Double clicks inside an APO grid. |
| **sapgui_apogrid_get_cell_data** | Gets the data from a specific APO grid cell. |
| **sapgui_apogrid_get_cell_format** | Gets the format of the specified APO grid cell. |
| **sapgui_apogrid_get_cell_tooltip** | Gets the tooltip of the specified APO grid cell. |
| **sapgui_apogrid_is_cell_changeable** | Checks whether the specified cell is editable. |
| **sapgui_apogrid_open_cell_context_menu** | Opens a context menu in the specified cell. |
| **sapgui_apogrid_press_ENTER** | Presses ENTER in an APO grid. |
| **sapgui_apogrid_scroll_to_column** | Scrolls to a specified column within the APO grid. |
| **sapgui_apogrid_scroll_to_row** | Scrolls to a specified row within the APO grid. |
| **sapgui_apogrid_select_all** | Selects all cells in the APO grid. |
| **sapgui_apogrid_select_cell** | Selects a cell in the APO grid. |
| **sapgui_apogrid_select_column** | Selects a column in the APO grid. |
| **sapgui_apogrid_select_row** | Selects a row in the APO grid. |
| **sapgui_apogrid_set_cell_data** | Sets the data in the specified APO grid cell. |

### Calendar Functions

| | |
|---|---|
| **sapgui_calendar_focus_date** | Sets the focus on a specific date. |
| **sapgui_calendar_scroll_to_date** | Scrolls to a specific date in the calendar. |
| **sapgui_calendar_select_interval** | Selects a range of dates within the calendar. |

### Grid Functions

| | |
|---|---|
| **sapgui_grid_clear_selection** | Clears a selection in a grid. |
| **sapgui_grid_click** | Clicks within a grid. |
| **sapgui_grid_click_current_cell** | Clicks within a grid's active cell. |
| **sapgui_grid_double_click** | Double-clicks within a grid. |
| **sapgui_grid_double_click_current_cell** | Double-clicks within a grid's active cell. |
| **sapgui_grid_get_cell_data** | Retrieves the text from a grid cell. |
| **sapgui_grid_get_current_cell_column** | Retrieves the KEY (Inner ID) of a column of the current cell within a grid. |
| **sapgui_grid_get_current_cell_row** | Retrieves the row number of a grid's current cell. |
| **sapgui_grid_is_checkbox_selected** | Checks the state of a check box in a grid. |
| **sapgui_grid_open_context_menu** | Right clicks in a grid to open a context menu. |
| **sapgui_grid_press_button** | Clicks a button in a grid cell. |
| **sapgui_grid_press_button_current_cell** | Clicks a button in the active grid cell. |
| **sapgui_grid_press_column_header** | Presses the column header of a grid. |

| | |
|---|---|
| **sapgui_grid_press_ENTER** | Presses ENTER within a grid. |
| **sapgui_grid_press_F1** | Presses F1 within a grid. |
| **sapgui_grid_press_F4** | Presses F4 within a grid. |
| **sapgui_grid_press_toolbar_button** | Clicks a toolbar button in a grid. |
| **sapgui_grid_press_toolbar_context_button** | Clicks a toolbar context button in a grid. |
| **sapgui_grid_press_total_row** | Clicks the total row area in a grid. |
| **sapgui_grid_press_total_row_current_cell** | Clicks the total row button in the currently active grid cell. |
| **sapgui_grid_scroll_to_row** | Scrolls to a row in a grid. |
| **sapgui_grid_select_cell** | Selects a cell in a grid. |
| **sapgui_grid_select_cell_column** | Selects a cell in the specified column of the current row. |
| **sapgui_grid_select_cell_row** | Selects a cell in the specified row of the current column. |
| **sapgui_grid_select_cells** | Selects cells in a grid. |
| **sapgui_grid_select_columns** | Selects columns in a grid. |
| **sapgui_grid_select_context_menu** | Selects a context menu in a grid. |
| **sapgui_grid_select_rows** | Selects rows in a grid. |
| **sapgui_grid_select_toolbar_menu** | Selects a toolbar menu in a grid. |
| **sapgui_grid_set_cell_data** | Inserts text into a grid cell. |
| **sapgui_grid_set_checkbox** | Selects or clears a grid check box. |
| **sapgui_grid_set_column_order** | Sets the column order in a grid. |

### Status Bar Functions

| | |
|---|---|
| **sapgui_status_bar_get_param** | Gets a parameter from the status bar. |
| **sapgui_status_bar_get_text** | Gets text from the status bar. |
| **sapgui_status_bar_get_type** | Retrieves status bar information: **Success**, **Warning**, or **Error**. |

### Table Functions

| | |
|---|---|
| **sapgui_table_is_checkbox_selected** | Checks the state of a check box in a table. |
| **sapgui_table_is_row_selected** | Checks if a table row is selected. |
| **sapgui_table_get_text** | Retrieves the text in a table cell. |
| **sapgui_table_is_radio_button_selected** | Checks the state of a radio button in a table. |
| **sapgui_table_press_button** | Presses a button in a table. |
| **sapgui_table_select_combobox_entry** | Selects a list entry within a table. |
| **sapgui_table_select_radio_button** | Selects a radio button in a table. |
| **sapgui_table_set_checkbox** | Selects or clears a check box in a table. |
| **sapgui_table_set_focus** | Sets the focus to a table. |
| **sapgui_table_set_password** | Sets the password within a table. |
| **sapgui_table_set_row_selected** | Selects or deselects a table row. |
| **sapgui_table_set_text** | Inserts text into a table cell. |

### Tree Functions

| | |
|---|---|
| **sapgui_tree_click_link** | Clicks a link in a tree. |
| **sapgui_tree_collapse_node** | Collapses a tree node. |
| **sapgui_tree_double_click_item** | Double-clicks a tree item. |
| **sapgui_tree_double_click_node** | Double-clicks a tree node. |

| | |
|---|---|
| **sapgui_tree_expand_node** | Expands a tree node. |
| **sapgui_tree_get_item_text** | Gets the text of a tree item. |
| **sapgui_tree_get_node_text** | Gets the text of a tree node. |
| **sapgui_tree_is_checkbox_selected** | Checks if a tree check box is selected. |
| **sapgui_tree_open_default_context_menu** | Opens a tree's default context sensitive menu. |
| **sapgui_tree_open_header_context_menu** | Opens a tree header's context sensitive menu. |
| **sapgui_tree_open_item_context_menu** | Opens a tree item's context sensitive menu. |
| **sapgui_tree_open_node_context_menu** | Opens a tree node's context sensitive menu. |
| **sapgui_tree_press_button** | Clicks a button in a tree. |
| **sapgui_tree_press_header** | Clicks on a column header in a tree. |
| **sapgui_tree_press_key** | Presses a key from within a tree. |
| **sapgui_tree_scroll_to_item** | Scrolls to a tree item. |
| **sapgui_tree_scroll_to_node** | Scrolls to a tree node. |
| **sapgui_tree_select_column** | Selects a column in a tree. |
| **sapgui_tree_select_item** | Selects an item in a tree. |
| **sapgui_tree_select_node** | Selects a node in a tree. |
| **sapgui_tree_set_checkbox** | Selects or clears a tree check box. |
| **sapgui_tree_set_column_width** | Sets the column width of a tree. |
| **sapgui_tree_set_hierarchy_header_width** | Sets the width of the tree hierarchy. |
| **sapgui_tree_set_selected_node** | Selects a node in a tree. |
| **sapgui_tree_unselect_all** | Cancels all selections in a tree. |

| | |
|---|---|
| **sapgui_tree_unselect_column** | Cancels the selection of a tree column. |
| **sapgui_tree_unselect_node** | Cancels the selection of a tree node. |

### Window Functions

| | |
|---|---|
| **sapgui_window_close** | Closes the SAPGUI client window. |
| **sapgui_window_maximize** | Sets window to full screen size. |
| **sapgui_window_resize** | Resizes a window to the specified size. |
| **sapgui_window_restore** | Restores the window to non-maximized state. |
| **sapgui_window_scroll_to_row** | Scrolls to a row in a window. |

### Verification and Data Retrieval Functions

| | |
|---|---|
| **sapgui_get_active_window_title** | Retrieves the active window's title. |
| **sapgui_get_ok_code** | Retrieves the Command field text. |
| **sapgui_get_text** | Gets the text from an object. |
| **sapgui_is_checkbox_selected** | Checks if a check box is selected. |
| **sapgui_is_object_available** | Checks whether an object is available. |
| **sapgui_is_object_changeable** | Checks if an object can be changed. |
| **sapgui_is_radio_button_selected** | Checks if a radio button is selected. |
| **sapgui_is_tab_selected** | Checks if a tab is selected. |

### General Object Functions

| | |
|---|---|
| **sapgui_htmlviewer_send_event** | Sends an event to the HTML Viewer. |
| **sapgui_select_combobox_entry** | Selects a list entry. |
| **sapgui_press_button** | Presses a button. |
| **sapgui_select_active_window** | Sets the specified window as the active window. |
| **sapgui_select_radio_button** | Selects a radio button. |
| **sapgui_select_tab** | Selects a tab. |
| **sapgui_send_vkey** | Sends a virtual key. |
| **sapgui_set_checkbox** | Selects or clears a check box. |
| **sapgui_set_focus** | Sets the focus to the specified object. |
| **sapgui_select_menu** | Selects the specified menu. |
| **sapgui_set_password** | Sets the text of the password field. |
| **sapgui_set_text** | Inserts text into a text box. |
| **sapgui_set_ok_code** | Sets the Command field text. |

## Tips for SAPGUI Vuser Scripts

The following sections provides Recording Tips, Replay Tips, and Tips for Replaying in a Scenario or Session Step for SAPGUI Vusers. In addition, you can obtain information directly from the SAP support site.

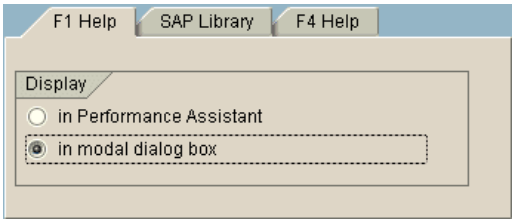
### Recording Tips

This section provides recording tips for a SAPGUI Vuser script.

➤ Make sure to record the actions into the appropriate sections: Record the logon procedure into the **vuser_init** section, the actions that you want to repeat in the **Actions** sections, and the logoff procedure in the **vuser_end**

section.

- When recording a multi-protocol script in which the SAPGUI client contains Web controls, close the SAPLogon application before recording.
- Use modal dialog boxes for F1. Instruct the SAPGUI client to open the F1 help in a modal dialog box. Choose **Help >Settings**. Click the **F1 Help** tab and select the **in modal dialog box** option in the Display section.



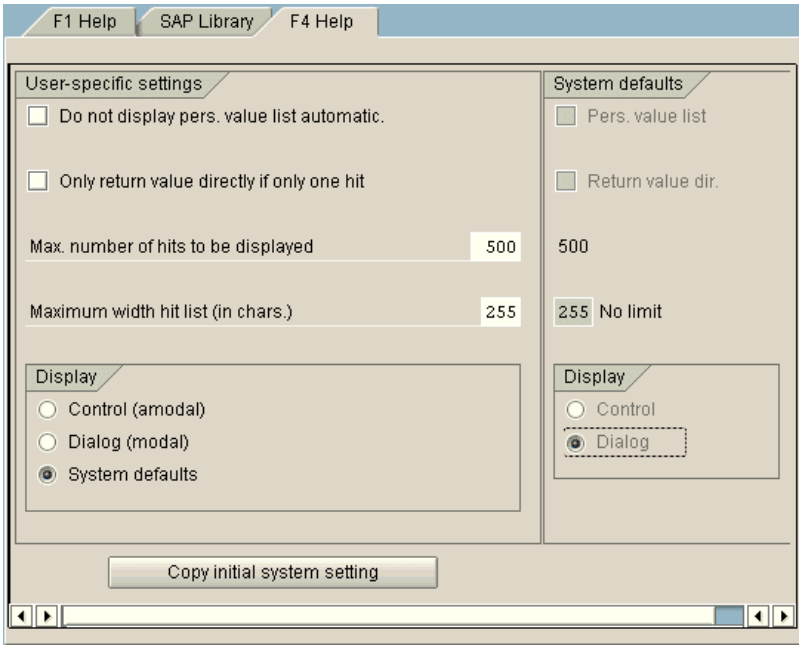

- Use modal dialog boxes for F4. Instruct the SAPGUI client to open the F4 help in a modal dialog box.

  The following procedure must be performed by a SAP administrator:

  **To open F4 help in modal dialog boxes:**

1. Ensure that all users have logged off from the server.

**2** Choose **Help** > **Settings**. Click the **F4 Help** tab.




**3** In the Display section (bottom left), choose System defaults.

**4** In the Display portion of the System defaults section (bottom right), select **Dialog**.

**5** Save the changes—click **Copy initial system setting** or CTRL+S.

**6** Verify that the status bar displays the message **Data was saved**.

**7** Close the session.

**8** Restart the service through the SAP Management Console.

### Replay Tips

Follow these guidelines before replaying your script in standalone-mode:

- Replace the encrypted password in the **sapgui_logon** function generated during recording, with the real password. It is the second argument of the function, after the user name: sapgui_logon("user", "pswd", "800", "EN"); For additional security, you can encrypt the password within the code. Select the password text (the actual text, not *****) and choose **Encrypt string** from the right-click menu. VuGen inserts an **lr_decrypt** function at the location of the password: sapgui_logon("user", lr_decrypt("3ea037b758"), "800", "EN");.
- When running a script for the first time, configure VuGen to show the SAPGUI user interface during replay, in order to see the operations being performed through the UI. To show the user interface during replay, open the run-time settings (F4) and select the **Show SAP Client During Replay** option in the **SAPGUI:General** node. During a load scenario, disable this option, since it uses a large amount of system resources in displaying the UI for multiple Vusers.

### Tips for Replaying in a Scenario or Session Step

The following sections provide configuration tips for running the script on a Controller or Console or Load Generator machine.

#### Controller and Console Settings

When working with a LoadRunner scenario or session step, set the following values when running your script in a load test configuration:

**Ramp-up:** One by one (to insure proper logon) in the Scheduler.

**Think time:** Random think time in the Run-Time settings.

**Users per load generator:** 50 Vusers for machine with 512 MB of memory in the Load Generators dialog box.

#### Load Generator Settings

When running your script in a scenario or session step, check the agent mode and configure the terminal sessions on the Load Generator machines.

**Agent Mode:** Make sure that the LoadRunner (or Performance Center) Remote Agent is running in Process mode. Service mode is not supported.

To check this, move your mouse over the agent's icon in the Windows task bar area, and read the description. If the description reads LoadRunner Agent Service, it is running as a service.




**To restart the agent as a process:**



1 Stop the agent. Right-click the LoadRunner Agent icon and select **Close**.

2 Run **magentproc.exe**, located in the **launch_service\bin** directory, under the LoadRunner or Tuning Module installation.

3 To ensure that the correct Agent is launched the next time you start your machine, change the Start type of the Agent Service from Automatic to Manual. Then add a shortcut to **magentproc.exe** to the Windows Startup folder.

**Terminal Sessions:** Machines running SAPGUI Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine. To increase the number of Vusers per machine, open additional terminal server sessions on the Load Generator machines. Choose **Agent Configuration** from **Start** > **Programs** > **Mercury** <**product_name**> > **Advanced Settings**, and select the Enable Terminal Service option. You specify the number of terminal sessions in the Load generator machine properties. For more information, see the Configuring Terminal Services section in the *LoadRunner Controller User's Guide*.

---

**Note:** When the LoadRunner Agent is running in a terminal session, and the terminal session's window is minimized, no snapshots will be captured on errors.

---

# Troubleshooting SAPGUI Vuser Scripts

**Question 1:** I was able to record a script, but why does replay fail?

**Answer:** In LoadRunner, make sure that the LoadRunner Remote Agent is running in Process mode. Service mode is not supported. For more information, see "Replay Tips" on page 940.

**Question 2:** Why were certain SAPGUI controls not recorded?

**Answer:** Some SAPGUI controls are only supported in their menu or toolbar contexts. Try performing the problematic task using a different means—through a menu option, context menu, toolbar, and so on.

**Question 3:** Why can't I record or replay any scripts in VuGen?

**Answer:**

1. Verify that you have the latest patch of SAPGUI 6.20 installed. The lowest allowed patch level is patch 32.
2. Make sure that scripting is enabled. See the "Checking the Configuration" on page 900.
3. Verify that notifications are disabled in the SAPGUI for Windows client. Click the Customizing of Local Layout button or press ALT+F12. Click **Options** and select the Scripting tab. Clear both **Notify** options.

**Question 4:** What is the meaning of the error popup messages that are issued when I try to run the script?

**Answer:** Certain SAP applications store the last layout for each user (such as which frames are visible or hidden). If the stored layout was changed since the script was recorded, this may cause replay problems. For Example, in the ME52N transaction, the "Document overview Off/On" button will change the number of visible frames.

**If this occurs:**

1. Navigate the transaction to the same point as it was during recording, before starting replay. You can use the Snapshot viewer to see the layout in which it was recorded.

**2** Add statements to the script that bring the transaction to the desired layout during replay. For example, if an optional frame interferes with your replay, insert a verification function that checks if the frame is open. If it is open, click a button to close it. For verification examples, see "Adding Verification Functions" on page 921.

**Question 5:** Can I use the single sign-on mechanism when running a script on a remote machine?

**Answer:** No, VuGen does not support the single sign-on connection mechanism. In your SAPGUI client, open the Advanced Options and clear the **Enable Secure Network Communication** feature. Note that you must re-record the script after you modify the Connection preferences.

**Question 6**: Can VuGen record all SAP objects?

**Answer:** Recording is not available for objects not supported by SAPGUI Scripting. See your recording log for information about those objects.

**Question 7**: Are all business processes supported?

**Answer:** VuGen does not support business processes that invoke Microsoft Office module controls, nor those that require the use of GuiXT. You can disable **GuiXT** from the SAPGUI for Windows client Options menu.

# Additional Resources

### LoadRunner and Tuning Module

For Online Help on dialog boxes, press F1 within a dialog box. You can also choose **Help** > **Contents and Index** to manually open the Help. In the Index tab, locate the **SAPGUI Vuser scripts** entry and click the appropriate sub-entry.

For Online Help with a function, click within the function or step, and click F1 to open the *Online Function Reference*.

### SAP

For more information, refer to the SAP website at www. sap.com or one of the following locations:

- **SAP Notes** - https://websmp103.sap-ag.de/notes

  Note #480149: New profile parameter for user scripting on the front end

  Note #587202: Drag & Drop is a known limitation of the SAPGUI interface

- **SAP Patches** - https://websmp104.sap-ag.de/patches

  SAP GUI for Windows - SAPGUI 6.20 Patch (the lowest allowed level is 32)

# 60

# Developing SAP-Web Vuser Scripts

You use VuGen's SAP-Web Vuser type, to record the activity in SAP Workplace or SAP Portal clients.

This chapter describes:

- About Developing SAP-Web Vuser Scripts
- Creating a SAP-Web Vuser Script
- Setting SAP-Web Recording Options
- Understanding a SAP-Web Vuser Script
- Replaying a SAP-Web Vuser Script

***The following information only applies to the SAP-Web protocol.***

## About Developing SAP-Web Vuser Scripts

You use VuGen to record typical SAP business processes. VuGen records SAP Workplace or Portal activity during the business processes, and generates a Vuser script. When you perform actions within your browser, VuGen generates functions that describe this activity. Each function begins with a **web** prefix.

During replay, these functions emulate user activity on the SAP Workplace or Portal clients. For example, **web_url** navigates to the PageBuilder.

```
web_url("PageBuilder[myPage]",
"URL=http://sonata.mercury.co.il/hrnp$30001/sonata.mercury.co.il:80/Acti
on/PageBuilder[myPage]?pageName=com.sapportals.pct.home.mynews",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.mercury.co.il/sapportal",
        "Snapshot=t2.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=/irj/services/laf/themes/portal/sap_mango_polarwind/..,
ENDITEM,
        LAST);
```

## Creating a SAP-Web Vuser Script

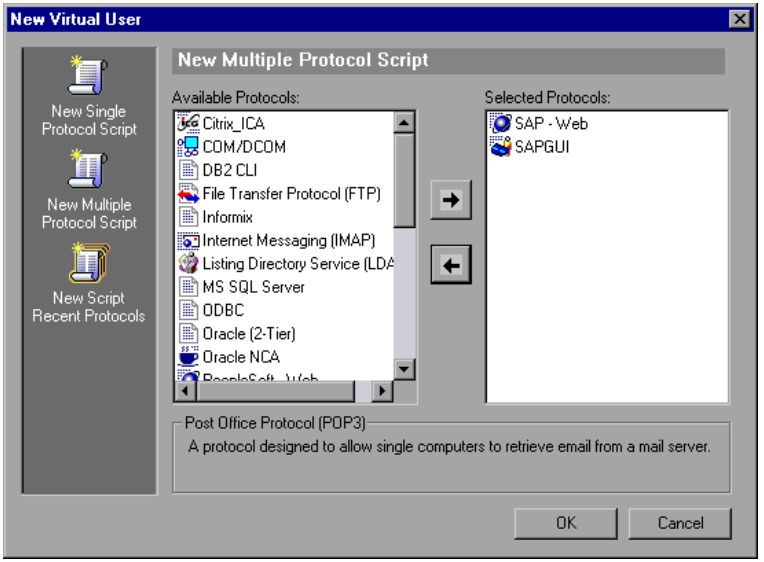
The first step in creating a SAP-Web Vuser script, is choosing the Vuser and script type. The **SAP-Web** Vuser is under the **ERP/CRM** category. You can create either a single or multi-protocol Vuser script. In addition, you can use the single-protocol SAPGUI/SAP-Web dual Vuser type.

**To create a SAP-Web Vuser:**

1. Invoke VuGen and choose **File** > **New**.
2. To record a session that does not incorporate any SAPGUI controls within the Workplace or Portal clients, create a single-protocol Vuser script using the **SAP-Web** Vuser type.

**3** To record a session that uses SAPGUI controls, create either:

- a single-protocol Vuser script, specifying the **SAPGUI/SAP-Web dual** protocol.
- a multi-protocol Vuser script, specifying both the **SAP-Web** and **SAPGUI** Vuser types.



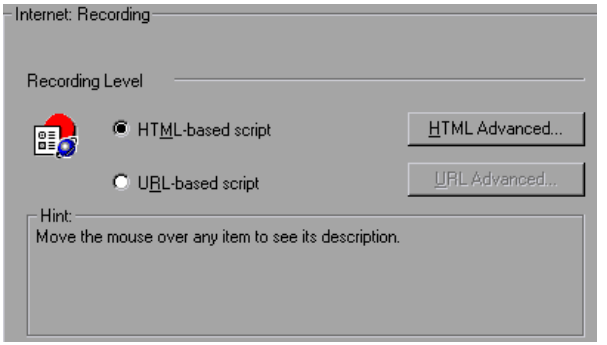
# Setting SAP-Web Recording Options

You use the recording options to set your preferences for how VuGen generates the Vuser script.

The recommended settings for the **Internet Protocol:Recording** node are:

**For SAP Workplace recordings:** URL-based script

**For SAP Portal recordings:** HTML-based script (the default)




For information about the other Web related recording options, see Chapter 40, “Setting Recording Options for Internet Protocols.”

## Understanding a SAP-Web Vuser Script

The SAP-Web Vuser script typically contains several SAP transactions which make up a business process. The business process consists of functions that emulate user actions. For information about these functions, see the Web functions in the *Online Function Reference* (**Help** > **Function Reference**).

The following example shows a typical recording for a SAP Portal client:

```
vuser_init()
{
	web_reg_find("Text=SAP Portals Enterprise Portal 5.0",
		LAST);

	web_set_user("junior{UserNumber}",
		lr_decrypt("3ed4cfe457afe04e"),
		"sonata.mercury.co.il:80");

	web_url("sapportal",
		"URL=http://sonata.mercury.co.il/sapportal",
		"Resource=0",
		"RecContentType=text/html",
		"Snapshot=t1.inf",
		"Mode=HTML",
		EXTRARES,
		"Url=/SAPPor-
tal/IE/Media/sap_mango_polarwind/images/header/branding_image.jpg",
"Referer=http://sonata.mercury.co.il/hrnp$30001/sonata.mer-
cury.co.il:80/Action/26011[header]", ENDITEM,
		"Url=/SAPPor-
tal/IE/Media/sap_mango_polarwind/images/header/logo.gif", "Ref-
erer=http://sonata.mercury.co.il/hrnp$30001/sonata.mercury.co.il:80/Actio
n/26011[header]", ENDITEM,
…
		LAST);
```

The following section illustrates a multi-protocol recording in which the Portal client opens a SAP control. Note the switch from **web_xxx** to **sapgui_xxx** functions.

```
web_url("dummy",
        "URL=http://sonata.mercury.co.il:1000/hrnp$30000/sonata.mer-
cury.co.il:1000/Action/dummy?PASS_PARAMS=YES&dummy-
Comp=dummy&Tcode=VA01&draggable=0&CompFName=VA01&Style=s
ap_mango_polarwind",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.mercury.co.il/sapportal",
        "Snapshot=t9.inf",
        "Mode=HTML",
        LAST);

    sapgui_open_connection_ex(" /H/Protector/S/3200 /WP",
        "",
        "con[0]");

    sapgui_select_active_connection("con[0]");

    sapgui_select_active_session("ses[0]");

    /*Before running script, enter password in place of asterisks in logon
function*/

    sapgui_logon("JUNIOR{UserNumber}",
        "ides",
        "800",
        "EN",
        BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui102",
        END_OPTIONAL);
```

# Replaying a SAP-Web Vuser Script

After creating and enhancing your SAP-Web Vuser script, you configure its run-time settings and run it from VuGen to check its functionality.

Run-Time settings let you control the Vuser behavior during replay. You configure these settings before running the Vuser script. You can set both General and Web related run-time settings.

The General settings include the run logic, pacing, logging, think time, and performance preferences. For information about the General run-time settings, see Chapter 12, "Configuring Run-Time Settings." For SAP-Web specific settings, see Chapter 42, "Configuring Internet Run-Time Settings."

Once you configure the Run-Time settings, you save the Vuser script and run it from VuGen as a standalone test, to verify that it runs correctly. For further information, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

After verifying that your Vuser script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# 61

# Developing Siebel-Web Vuser Scripts

You use VuGen to record the activity in a Siebel Web environment and generate a Vuser script. When you run the script, Vusers emulate the actions within your Siebel environment.

This chapter describes:

- About Developing Siebel-Web Vuser Scripts
- Recording a Siebel-Web Session
- Correlating Siebel-Web Scripts
- Correlating SWECount, ROWID, and SWET Parameters
- Troubleshooting Siebel-Web Vuser Scripts

***The following information only applies to Siebel-Web Vuser scripts.***

## About Developing Siebel-Web Vuser Scripts

The Siebel-Web protocol is similar to the standard Web Vuser, with several changes in the default settings to allow it to work with the Siebel Customer Relationship Management (CRM) application.

You record typical activities in your Siebel session. VuGen records the actions and generates functions with a **web_** prefix, that emulate the actions.

The sections below provide tips for working with Siebel-Web recorded Vuser Scripts and provide samples of the parameters that need to be correlated.

# Recording a Siebel-Web Session

When recording a Siebel-Web session, use the following guidelines:

**To record a Siebel-Web Vuser script:**

1. Create a Siebel-Web type Vuser script from the ERP category.
2. Set the following Recording Options:
   - Record node: **HTML based script**

     Advanced HTML - Script options: **a script containing explicit URLs only**

     Advanced HTML - Non HTML-generated elements: **Do not record**
   - Advanced node: Clear the **Reset context for each action** option.
3. Record the login in the **vuser_init** section.
4. Record the Business Process in **Action1**.
5. Record the logout in the **vuser_end** section.
6. In the Run-Time settings, clear the **Simulate a new user on each iteration** option in the Browser Emulation node.

For more information on how to configure the Recording Options and Web related Run-Time settings, see Chapter 40, "Setting Recording Options for Internet Protocols", and Chapter 42, "Configuring Internet Run-Time Settings."

# Correlating Siebel-Web Scripts

When creating a test script for a Siebel session, you will most probably need to use correlation in your script. Correlation is the mechanism by which VuGen saves dynamic values to parameters during record and replay, for use at a later point in the script. If you replayed the recorded script as is, without correlation, it would fail, since the values of the arguments differ each time the script runs. An example of such variables are **SWECount** and **SWEBMC**.

When you use correlation, VuGen saves the dynamic variables during both record and replay, and uses them at the appropriate points within the script. You can instruct VuGen to apply correlation during recording using one of the following methods:

- Siebel Correlation Library

  The Siebel correlation library automatically correlates most of the dynamic values, creating a concise script that you can replay without major modifications. This is the recommended method for correlation.

- VuGen Native Siebel Correlation

  The native, built-in rules, work on a low level, allowing you to debug your script and understand the correlations in depth.

## Siebel Correlation Library

To assist you with correlation, Siebel has released a correlation library file as part of the Siebel Application Server version 7.7. This library is available only through Siebel. The library file, **ssdtcorr.dll**, is located under the siebsrvr\bin folder for Windows and under siebsrvr/lib for UNIX installations.

The library file, **ssdtcorr.dll**, must be available to all machines where a Load Generator, Controller, or Console reside. Support for this library requires VuGen 8.0 and higher.

**To enable correlation with this library:**

1. Copy the DLL file into the bin directory of the Mercury product installation.
2. Open a multi-protocol script using the **Siebel-Web** Vuser type.

3 Enable UTF-8 support in the recording options. For more information, see "Setting Advanced Recording Options" on page 564.

4 Open the recording option's Correlation node and click **Import**. Import the rules file, **WebSiebel77Correlation.cor**, from the \dat\webrulesdefaultsetting directory. If you are prompted with warnings, click **Override**. For more information, see "Setting the Correlation Recording Options" on page 673.

To revert back to the default correlation, delete all of the Siebel rules and click **Use Defaults**.

When using the Siebel correlation library, verify that the SWE count rules (where the left boundary contains the **SWEC** string) are not disabled. For more information, see "Disabling and Enabling Rules" on page 960.

### VuGen Native Siebel Correlation

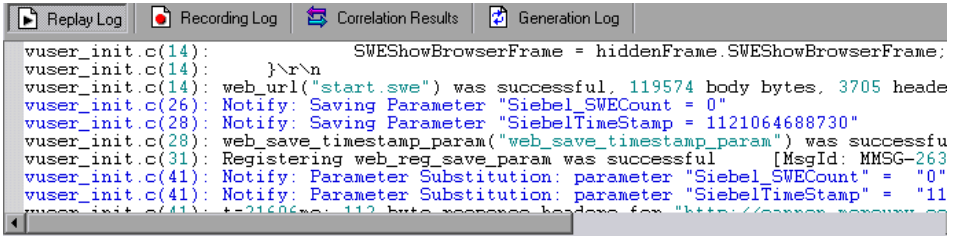
VuGen's native built-in rules for the Siebel server detect the Siebel server variables and strings, automatically saving them for use at a later point within the script.

You can view these rules in the list of correlation rules (see "Using VuGen's Correlation Rules" on page 664). The rules list the boundary criteria that are unique for Siebel server strings.

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

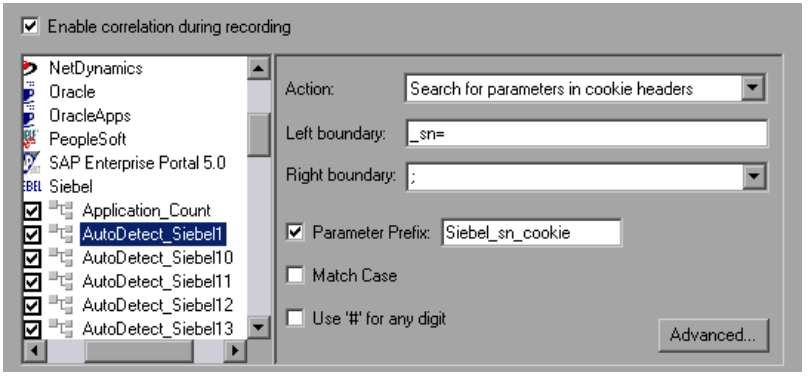
You can also create your own rules by entering unique boundary criteria in the Correlation Recording Options (see Chapter 46, "Setting Correlation Rules for Web Vuser Scripts") or after the recording from the Correlation Results tab (see "Performing a Scan for Correlations" on page 683).

In the Replay Log tab, VuGen indicates when it registers, saves, or uses the parameters. Note that to display this information, you need to enable Extended logging. For more information, see “Configuring the Log Run-Time Settings” on page 184.




### Simple Variable Correlation

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



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

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

…

web_url("start.swe_3",
"URL=http://cannon.mercury.co.il/callcenter_enu/start.swe?SWECmd=Got
oPostedAc-
tion&SWEDIC=true&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECo
unt}&SWEFrame=top._sweclient&SWECS=true",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://cannon.mercury.co.il/callcenter_enu/start.swe?SWECmd=
GetCached-
Frame&_sn={Siebel_sn_cookie6}&SWEC={Siebel_SWECount}&SWE-
Frame=top._swe",
"Snapshot=t4.inf",
"Mode=HTML",
LAST);
```

### Function Correlation

In certain instances, the boundary match is a function. Functions generally use an array to store the run-time values. In order to correlate these values, VuGen parses the array and saves each argument to a separate parameter using the following format:

```
<parameter_name> = <recorded_value> (display_name)
```

The display name is the text that appears next to the value, in the Siebel Application.

VuGen inserts a comment block with all of the parameter definitions.

```
/* Registering parameter(s) from source task id 159
      // {Siebel_Star_Array_Op33_7} = ""
      // {Siebel_Star_Array_Op33_6} = "1-231"
      // {Siebel_Star_Array_Op33_2} = ""
      // {Siebel_Star_Array_Op33_8} = "Opportunity"
      // {Siebel_Star_Array_Op33_5} = "06/26/2003 19:55:23"
      // {Siebel_Star_Array_Op33_4} = "06/26/2003 19:55:23"
      // {Siebel_Star_Array_Op33_3} = ""
      // {Siebel_Star_Array_Op33_1} = "test camp"
      // {Siebel_Star_Array_Op33_9} = ""
      // {Siebel_Star_Array_Op33_rowid} = "1-6F"
      // */
```

In addition, when encountering a function, VuGen generates a new parameter for **web_reg_save_param**, AutoCorrelationFunction. VuGen also determines the prefix of the parameters and uses it as the parameter name. In the following example, the prefix is **Siebel_Star_Array_Op33**.

```
web_reg_save_param("Siebel_Star_Array_Op33",
      "LB/IC=`v`",
      "RB/IC=`",
      "Ord=1",
      "Search=Body",
      "RelFrameId=1",
      "AutoCorrelationFunction=flCorrelationCallbackParseStarArray",
      LAST);
```

VuGen uses the parameters at a later point within the script. In the following example, the parameter is called in **web_submit_data**.

```
web_submit_data("start.swe_14",
  "Action=http://cannon.mercury.co.il/callcenter_enu/start.swe",
  "Method=POST",
  "RecContentType=text/html",
  "Referer=",
  "Snapshot=t15.inf",
  "Mode=HTML",
  ITEMDATA,
  "Name=SWECLK", "Value=1", ENDITEM,
  "Name=SWEField", "Value=s_2_1_13_0", ENDITEM,
  "Name=SWER", "Value=0", ENDITEM,
  "Name=SWESP", "Value=false", ENDITEM,
  "Name=s_2_2_29_0", "Value={Siebel_Star_Array_Op33_1}",
ENDITEM,
  "Name=s_2_2_30_0", "Value={Siebel_Star_Array_Op33_2}",
ENDITEM,
  "Name=s_2_2_36_0", "Value={Siebel_Star_Array_Op33_3}",
ENDITEM,
…
```

During replay, Vusers do a callback to the public function, using the array elements that were saved as parameters.

---

**Note:** Correlation for the **SWEC** parameter is not done through the correlation rules. VuGen handles it automatically with a built-in detection mechanism. For more information, see "SWEC Correlation" on page 961.

---

### Disabling and Enabling Rules

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

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

**To disable rules:**

1. Open the Correlation recording options. Choose **Tools** > **Recording Options** and click the Correlation node.
2. Select the **Enable correlation during recording** option. The dialog box displays the supported servers.
3. Expand the rules under Siebel and view their properties.
4. Clear the check box adjacent to the rule for each rule you want to disable.

### SWEC Correlation

SWEC is a parameter used by Siebel servers representing the number of user clicks. The SWEC parameter usually appears as an argument of a URL or a POST statement. For example:

```
GET /callcenter_enu/start.swe?SWECmd=GetCachedFrame&_sn=2-
mOTFXHWBAAGb5Xzv9Ls2Z45QvxGQnOnPVtX6vnfUU_&SWEC=1&S
WEFrame=top._swe._sweapp HTTP/1.1
```

or

```
POST /callcenter_enu/start.swe HTTP/1.1
…
\r\n\r\n
SWERPC=1&SWEC=0&_sn=2-
mOTFXHWBAAGb5Xzv9Ls2Z45QvxGQnOnPVtX6vnfUU_&SWECmd=In
vokeMethod…
```

VuGen handles the changes of the SWEC by incrementing a counter before each relevant step. VuGen stores the current value of the SWEC in a separate variable (Siebel_SWECount_var). Before each step, VuGen saves the counter's value to a VuGen parameter (Siebel_SWECount).

In the following example, **web_submit_data** uses the dynamic value of the SWEC parameter, Siebel_SWECount.

```
Siebel_SWECount_var += 1;

lr_save_int(Siebel_SWECount_var, "Siebel_SWECount");

web_submit_data("start.swe_8",
      "Action=http://cannon.mercury.co.il/callcenter_enu/start.swe",
      "Method=POST",
      "TargetFrame=",
      "RecContentType=text/html",
      "Referer=",
      "Snapshot=t9.inf",
      "Mode=HTML",
      "EncodeAtSign=YES",
      ITEMDATA,
      "Name=SWERPC", "Value=1", ENDITEM,
      "Name=SWEC", "Value={Siebel_SWECount}", ENDITEM,
      "Name=SWECmd", "Value=InvokeMethod", ENDITEM,
      "Name=SWEService", "Value=SWE Command Manager", ENDITEM,
      "Name=SWEMethod", "Value=BatchCanInvoke", ENDITEM,
      "Name=SWEIPS",…
      LAST);
```

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

## Correlating SWECount, ROWID, and SWET Parameters

This section provides tips for correlating several special parameters:

- SWECount
- Row ID Length
- SWETS (Timestamps)

### SWECount

The SWECount parameter value is usually a small number consisting of one or two digits. It is often difficult to determine where to replace the recorded value with a parameter.

In the **web_submit_data** function, VuGen only replaces it in the SWEC field.

In URLs, VuGen only replaces the value when it appears after the strings "SWEC=" or "SWEC`".

The parameter name for all the SWECount correlations is the same.

### Row ID Length

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

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

If you define parameters to correlate the string as

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

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

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

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

### SWETS (Timestamps)

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

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

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

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

## Troubleshooting Siebel-Web Vuser Scripts

This section provides information about errors you might encounter when creating a script, and the breakdown diagnostic tool.

- Typical Errors
- Recording Breakdown Information

### Typical Errors

You may encounter one or more of the following errors while creating a Siebel-Web Vuser script:

- Back or Refresh Error
- Same Values
- No Content HTTP Response
- Restoring the Context
- Cannot Locate Record
- End of File
- Unable to Retrieve Search Categories

#### Back or Refresh Error

An error message relating to **Back or Refresh** typically has the following text:

We are unable to process your request. This is most likely because you used the browser BACK or REFRESH button to get to this point.

**Cause:** The possible causes of this problem may be:

- The SWEC was not correlated correctly for the current request.
- The SWETS was not correlated correctly for the current request.
- The request was submitted twice to the Siebel server without the SWEC being updated.

➤ A previous request should have opened a frame for the browser to download. This frame was not created on the server probably because the SWEMethod has changed since the recording.

### Same Values

A typical Web page response to the **Same Values** error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`10`0`1`Errors`0`2`0`Level0`0`ErrMsg`Th
e same values for 'Name' already exist. If you would like to enter a new record,
please ensure that the field values are unique.`ErrCode`28591`
```

**Cause:** The possible cause of this problem may be that one of the values in the request (in the above example, the value of the Name field) duplicates a value in another row of the database table. This value needs to be replaced with a unique value to be used for each iteration per user. The recommended solution is to replace the row ID with its parameter instead insuring that it will be unique.

### No Content HTTP Response

A typical HTTP response for a **No Content HTTP Response** type error is:

```
HTTP/1.1 204 No Content
Server: Microsoft-IIS/5.0
Date: Fri, 31 Jan 2003 21:52:30 GMT
Content-Language: en
Cache-Control: no-cache
```

**Cause:** The possible causes of this problem may be that the row ID is not correlated at all or that it is correlated incorrectly.

### Restoring the Context

The typical Web page response to the **Restoring the Context** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`9`0`1`Errors`0`2`0`Level0`0`ErrMsg`An
error happened during restoring the context for requested
location`ErrCode`27631`
```

**Cause:** The possible causes of this problem may be that the rowid is not correlated or that it is correlated incorrectly.

### Cannot Locate Record

The typical Web page response to the **Cannot locate record** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`23`0`2`Errors`0`2`0`Level0`0`ErrMsg`Ca
nnot locate record within view: Contact Detail - Opportunities View applet:
Opportunity List Applet.`ErrCode`27573`
```

**Cause:** The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

### End of File

The typical Web page response to the **End of File** type error is:

```
@0`0`3`3``0`UC`1`Status`Error`SWEC`28`0`1`Errors`0`2`0`Level0`0`ErrMsg`An
end of file error has occurred. Please continue or ask your systems administrator
to check your application configuration if the problem persists.`ErrCode`28601`
```

**Cause:** The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

### Unable to Retrieve Search Categories

The typical Web page response to the **Unable to Retrieve Search Categories** type error is:

**Cause:** A possible cause of this problem may be that the search frame was not downloaded from the server. This occurs when the previous request did not ask the server to create the search frame correctly.

## Recording Breakdown Information

VuGen provides a diagnostic tool for understanding the transaction components in your test—**transaction breakdown**. Using transaction breakdown, you can determine where the bottlenecks are and the issues that need to be resolved.

When preparing your script for transaction breakdown, it is recommended that you add think time at the end of each transaction using the ratio of one second per hour of testing. For more information on entering think time, see Chapter 7, "Enhancing Vuser Scripts."

In order to record the transaction breakdown information, you need to modify your the parameterization functions in your script.

**To prepare your script for transaction breakdown:**

**1** Identify the script parameterization replacement of the Session ID.

```
/* Registering parameter(s) from source task id 15
    // {Siebel_sn_body4} = "28eMu9uzkn.YGFFevN1FdrCfCCOc8c_"
    // */
    web_reg_save_param("Siebel_sn_body4",
        "LB/IC=_sn=",
        "RB/IC=&",
        "Ord=1",
        "Search=Body",
        "RelFrameId=1",
        LAST);
```

**2** Mark the next **web_submit_data** function as a transaction by enclosing it with **lr_start_transaction** and **lr_end_transaction** functions.

**3** Before the end of the transactions, add a call to **lr_transaction_instance_add_info**, where the first parameter, 0 is mandatory and the session ID has a SSQLBD prefix.

```
lr_start_transaction("LoginSQLSync");
    web_submit_data("start.swe_2",
        "Action=http://design/callcenter_enu/start.swe",
        "Method=POST",
        "RecContentType=text/html",
        "Referer=http://design/callcenter_enu/start.swe",
        "Snapshot=t2.inf",
        "Mode=HTML",
        ITEMDATA,
        "Name=SWEUserName", "Value=wrun", ENDITEM,
        "Name=SWEPassword", "Value=wrun", ENDITEM,
        "Name=SWERememberUser", "Value=Yes", ENDITEM,
        "Name=SWENeedContext", "Value=false", ENDITEM,
        "Name=SWEFo", "Value=SWEEntryForm", ENDITEM,
        "Name=SWETS", "Value={SiebelTimeStamp}", ENDITEM,
        "Name=SWECmd", "Value=ExecuteLogin", ENDITEM,
        "Name=SWEBID", "Value=-1", ENDITEM,
        "Name=SWEC", "Value=0", ENDITEM,
        LAST);

lr_transaction_instance_add_info(0,lr_eval_string("SSQLBD:{Siebel_sn_body4}
"));
lr_end_transaction("LoginSQLSync", LR_AUTO);
```

---

**Note:** To avoid session ID conflicts, make sure that the Vusers log off from the database at the end of each session.

---

# Part XI

## Legacy Protocols

# 62

# Introducing RTE Vuser Scripts

RTE Vusers operate terminal emulators in Windows environments. This chapter describes how to develop Windows-based RTE Vuser scripts.

This chapter describes:

- About Developing RTE Vuser Scripts
- 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 test client/server systems.

You record a terminal emulator session with VuGen to represent a true user's actions. You can then enhance your recorded script with transaction and synchronization functions.

This chapter describes how to develop Windows-based RTE Vuser scripts.

# Introducing RTE Vusers

An RTE Vuser types character input into a terminal emulator, submits the data to a server, and then waits for the server to respond. For instance, suppose that you have a server that maintains customer information for a maintenance company. Every time a field service representative makes a repair, he accesses the server database by modem using a terminal emulator. The service representative accesses information about the customer and then records the details of the repair that he performs.

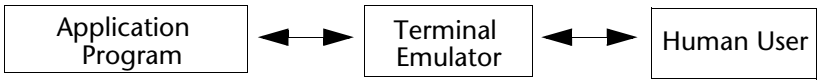
You could use RTE Vusers to emulate this case. An RTE Vuser would:

1. Type **60** at the command line to open an application program.
2. Type **F296**, the field service representative's number.
3. Type **NY270**, the customer number.
4. Wait for the word "Details" to appear on the screen. The appearance of "Details" indicates that all the customer details are displayed on the screen.
5. Type **Changed gasket P249, and performed Major Service** the details of the current repair.
6. Type **Q** to close the application program.

You use VuGen to create RTE Vuser scripts. The script generator records the actions of a human user in a terminal emulator. It records the keyboard input from the terminal window, generates the appropriate statements, and inserts them into the Vuser script. While you record, the script generator automatically inserts synchronization functions into the script. For details, see Chapter 65, "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 PowerTerm, a terminal emulator.




PowerTerm 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 63, “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 7, “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 8, "Working with VuGen Parameters."

**4 Configure the run-time settings.**

The run-time settings control the Vuser behavior during script execution. These settings include loop, log, and timing information.

For details, see Chapter 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you successfully create an RTE script, you integrate it into a scenario, profile, or session step. For more information on integrating scripts in a scenario, profile, or session step, refer to the appropriate 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_line_attribute** | Returns information about text formatting. |
| **TE_get_text_line** | Reads text from a designated line on the screen. |

### Cursor Functions

| | |
|---|---|
| **TE_get_cursor_pos** | Returns the current location of the cursor. |
| **TE_set_cursor_pos** | Sets the position of the cursor on the terminal screen. |

### System Variable Functions

| | |
|---|---|
| **TE_getvar** | Returns the value of an RTE system variable. |
| **TE_setvar** | Sets the value of an RTE system variable. |

### Error Code Functions

| | |
|---|---|
| **TE_perror** | Prints an error code to the Output window. |
| **TE_sperror** | Translates an error code into a string. |

### Typing Functions

| | |
|---|---|
| **TE_type** | Sends a formatted string to the client application. |
| **TE_typing_style** | Determines the way text is typed into the terminal emulator. |
| **TE_unlock_keyboard** | Unlocks the keyboard of a mainframe terminal. |

### 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_sync_transaction** | Records the time that the system remained in the most recent X SYSTEM mode. |
| **TE_wait_text** | Waits for a string to appear in a designated location. |

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 8, "Working with VuGen Parameters."

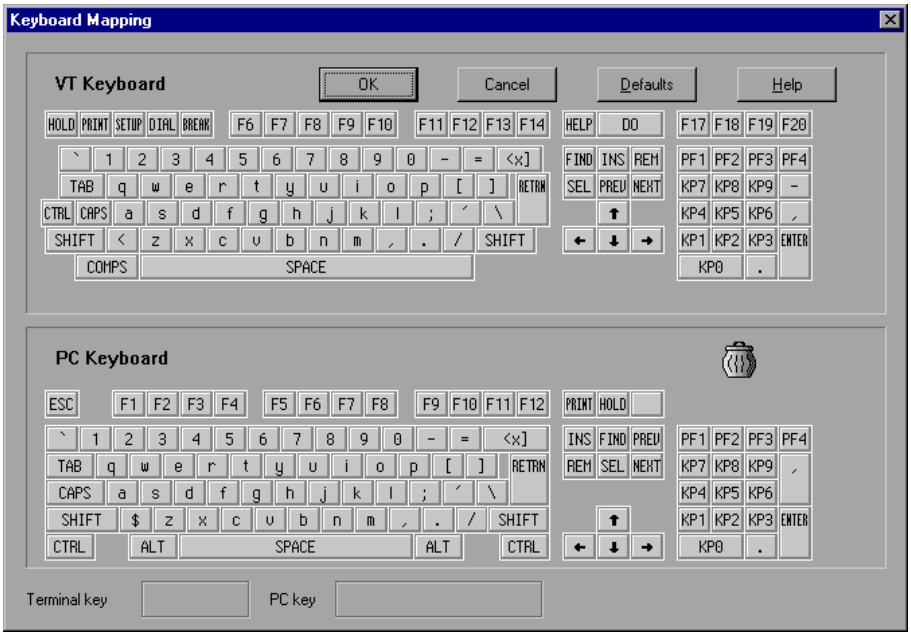
# Mapping Terminal Keys to PC Keyboard Keys

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**.

# 63

# Recording RTE Vuser Scripts

You use VuGen to record Windows-based Remote Terminal Emulation (RTE) Vuser scripts.

This chapter describes:

- About Recording RTE Vuser Scripts
- Creating a New RTE Vuser Script
- Recording the Terminal Setup and Connection Procedure
- Recording Typical User Actions
- Recording the Log Off Procedure
- Setting the RTE 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

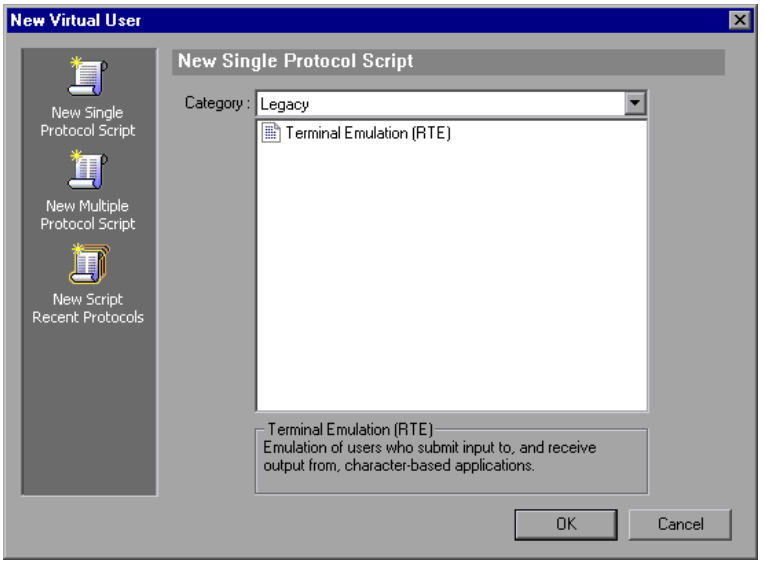
Before recording a user's actions into a Vuser script, you must open one. You can open an existing script, or create a new one. You use VuGen to create a new Vuser script.

**To create a new RTE Vuser script:**

1 Select **Virtual User Generator** from your product's start menu. The VuGen window opens.



**2** Click the **New** button. The **New Virtual User** dialog box opens:




**3** Select the **Legacy** protocol type, and 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.

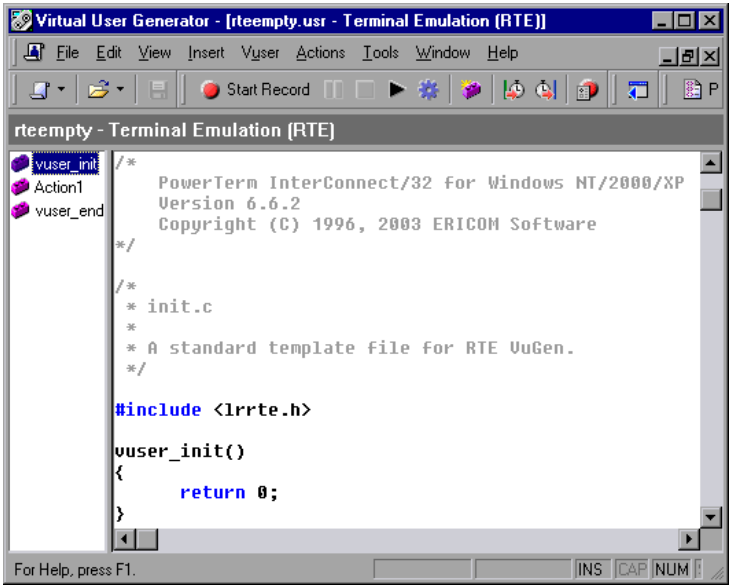
## Recording the Terminal Setup and Connection Procedure

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 12, "Configuring Run-Time Settings."

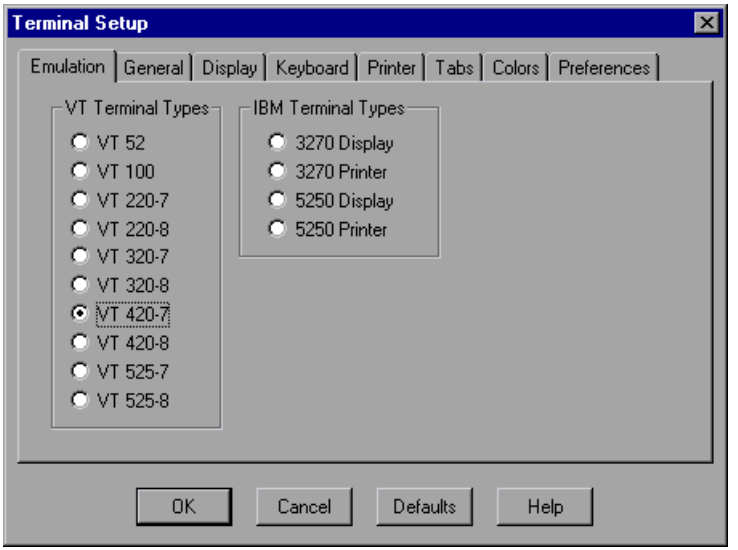
---

**3** In the Vuser script, place the cursor at the location where you want to begin recording.



**4** Click the **Record** button. The PowerTerm main window opens.

**5** From the PowerTerm menu bar, select **Terminal** > **Setup** to display the Terminal Setup dialog box.



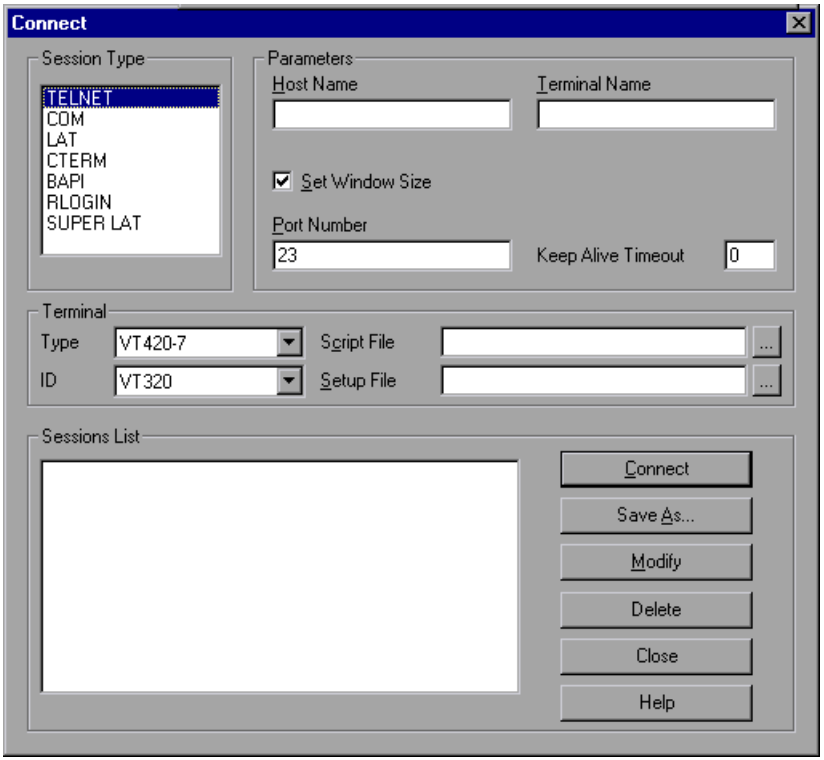

**6** Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.

---

**Note:** Select an IBM terminal type to connect to an AS/400 machine or an IBM mainframe; select a VT terminal type to connect to a UNIX workstation.

---

**7** Select **Communication** > **Connect** to display the Connect dialog box.




**8** Under **Session Type**, select the type of communication to use.

**9** Under **Parameters**, specify the required options. The available parameters vary depending on the type of session that you select. For details on the parameters, click **Help**.

---

**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 **TE_connect** statement has the following form:

```
/* *** The terminal type is VT220-7. */
 TE_connect(
  "comm-type = telnet;"
  "host-name = pharaoh;"
  "set-window-size = true;"
  "security-type = unsecured;"
  "telnet-binary-mode = true;"
  "terminal-type = vt220-7;"
  "terminal-model = vt320;"
  , 60000);
 if (TE_errno != TE_SUCCESS)
  return -1;
```

The inserted **TE_connect** statement is followed by an if statement that checks whether or not the **TE_connect** function succeeds during replay.

---

**Note:** Do not record more than one connection to a server **(TE_connect)** in a Vuser script.

---

The terminal setup and connection procedure is complete. You are now ready to begin recording typical user actions into the Vuser script, as described below.

## Recording Typical User Actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the **Actions** section of the Vuser script. Only the **Actions** section of a Vuser script is repeated when you run multiple iterations of the script. For details on setting iterations, see Chapter 12, "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 RTE Recording Options" on page 989.

---

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 12, "Configuring Run-Time Settings."

**To record the log off procedure:**

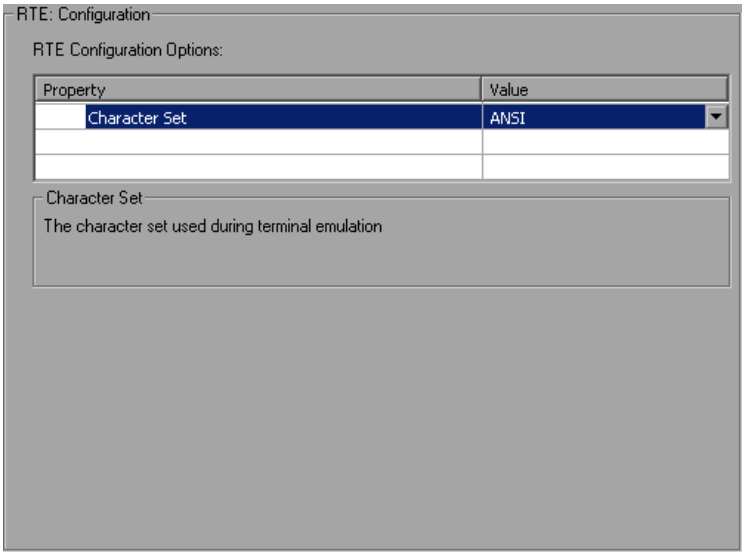
1. Ensure that you have performed and recorded the typical user actions as described in the previous section.
2. In the VuGen main window, click **vuser_end** in the **Section** box.
3. Perform the log off procedure. VuGen records the procedure into the **vuser_end** section of the script.
4.  Click **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 RTE Configuration Options

You can set the recording options to match the character set used during terminal emulation. The default character set is ANSI. For Kanji and other multi-byte platforms, you can specify DBCS (Double-byte Character Set).



To open the Configuration Recording Options, click the **Recording Options** button on the toolbar or select **Tools** > **Recording Options**. Select the **RTE:Configuration** node.

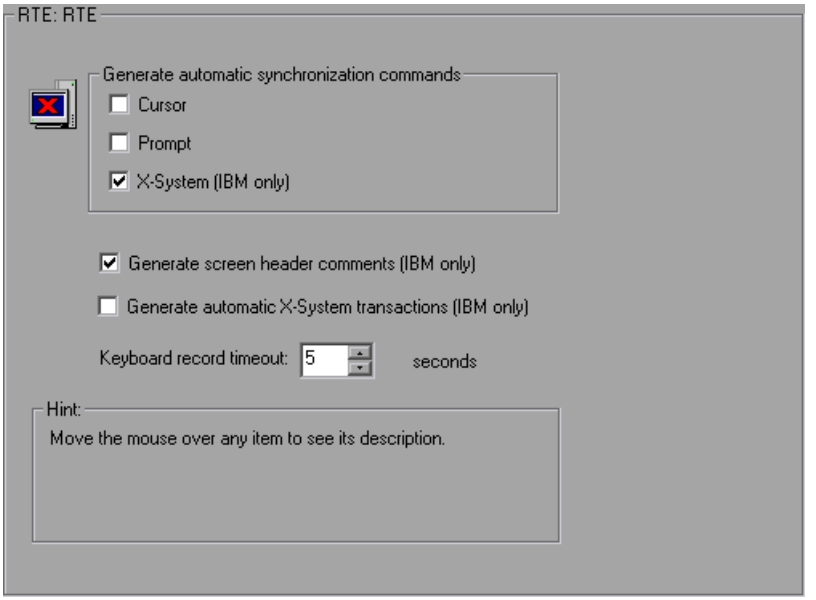


# Setting the RTE Recording Options

By setting the recording options, you can customize the code that VuGen generates for RTE functions. You use the Recording Options dialog box to set the recording options.



To open the Recording Options dialog box, click the **Recording Options** button on the toolbar, or select **Tools** > **Recording Options**. Select the **RTE:RTE** node.




You can set the following recording options:

- Automatic Synchronization Commands
- Automatic Screen Header Comments (IBM terminals only)
- Automatic X-System Transactions (IBM terminals only)
- Keyboard Recording Timeout

### Automatic Synchronization Commands

VuGen can automatically generate a number of TE-synchronization functions, and insert them into the script while you record.

**1** You can specify that VuGen generate a **TE_wait_sync** function each time a new screen is displayed while recording. To do so, select the **X-System** check box in the Recording Options dialog box.

By default, VuGen does automatically generate a **TE_wait_sync** function each time a new screen is displayed while recording.

---

**Note:** VuGen generates **TE_wait_sync** functions when recording IBM block mode terminals only.

---

**2** You can specify that VuGen generate a **TE_wait_cursor** function before each **TE_type** function. To do so, select the **Cursor** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate **TE_wait_cursor** functions.

**3** You can specify that VuGen generate a **TE_wait_text** function before each **TE_type** function (where appropriate). To do so, select the **Prompt** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate a **TE_wait_text** function before each **TE_type** function.

---

**Note:** VuGen generates meaningful **TE_wait_text** functions when recording VT type terminals only. Do not use automatic **TE_wait_text** function generation when recording block-mode (IBM) terminals.

---

### Automatic Screen Header Comments (IBM terminals only)

You can instruct VuGen to automatically generate screen header comments while recording a Vuser script, and insert the comments into the script.

Generated comments make a recorded script easier to read by identifying each new screen as it is displayed in the terminal emulator. A generated comment contains the text that appears on the first line of the terminal emulator window. The following generated comment shows that the Office Tasks screen was displayed in the terminal emulator:

```
/* OFCTSK                Office Tasks                */
```

To 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 or scenario run. To do so, VuGen inserts a **TE_wait_sync_transaction** function after each **TE_wait_sync** function. Each **TE_wait_sync_transaction** function creates a transaction with the name "default". Each **TE_wait_sync_transaction** function records the time that the system spent in the previous X SYSTEM state.

To instruct VuGen to insert **TE_wait_sync_transaction** statements while recording, select the **Generate automatic X SYSTEM transactions** check box in the Recording Options dialog box.

By default, VuGen does not automatically generate transactions.

### Keyboard Recording Timeout

When you type text into a terminal emulator while recording, VuGen monitors the text input. After each keystroke, VuGen waits up to a specified amount of time for the next key stroke. If there is no subsequent keystroke within the specified time, VuGen assumes that the command is complete. VuGen then generates and inserts the appropriate **TE_type** function into the script.

To set the maximum amount of time that VuGen waits between successive keystrokes, enter an amount in the **Keyboard record timeout** box.

By default, VuGen waits a maximum of 5 seconds between successive keystrokes before generating the appropriate **TE_type** function.

## Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to "type" character input into the PowerTerm terminal emulator:

- **TE_type** sends characters to the terminal emulator. When recording, the VuGen automatically generates **TE_type** functions for keyboard input to the terminal window. For details, see "Using the TE_type Function" on page 993.
- **TE_typing_style** determines the speed at which the Vuser types. You can manually define the typing style by inserting a **TE_typing_style** function into the Vuser script. For details, see "Setting the Typing Style" on page 994. Alternatively, you can set the typing style by using the run-time settings. For details, see Chapter 64, "Configuring RTE Run-Time Settings."

---

**Note:** While recording an RTE Vuser script, do not use the mouse to relocate the cursor within the terminal emulator window. VuGen does not record these cursor movements.

---

## Using the TE_type Function

When you record a script, the VuGen records all keyboard input and generates appropriate **TE_type** functions. During execution, **TE_type** functions send formatted strings to the terminal emulator.

Keyboard input is defined as a regular text string (including blank spaces). For example:

```
TE_type ("hello, world");
```

Input key names longer than one character are represented by identifiers beginning with the letter *k*, and are bracketed within greater-than/less-than signs (< >).

For example, the following function depicts the input of the Return key followed by the Control and y keys:

```
TE_type("<kReturn><kControl-y>");
```

Some other examples include: <kF1>, <kUp>, <kF10>, <kHelp>, <kTab>.

To determine a key name, record an operation on the key, and then check the recorded statement for its name.

---

**Note:** When you program a **TE_type** statement (rather than recording it), use the key definitions provided in the *Online Function Reference* (**Help > Function Reference**).

---

### Setting the Timeout Value for TE_type

If a Vuser attempts to submit a **TE_type** statement while the system is in X SYSTEM (or input inhibited) mode, the Vuser will wait until the X SYSTEM mode ends before typing. If the system stays in X SYSTEM mode for more than TE_XSYSTEM_TIMEOUT milliseconds, then the **TE_type** function returns a TE_TIMEOUT error.

You can set the value of TE_XSYSTEM_TIMEOUT by using **TE_setvar**. The default value for TE_XSYSTEM_TIMEOUT is 30 seconds.

### Allowing a Vuser to Type Ahead

Under certain circumstances you may want a Vuser to submit a keystroke even though the system is in X SYSTEM (or input inhibited) mode. For example, you may want the Vuser to press the Break key. You use the TE_ALLOW_TYPEAHEAD variable to enable the Vuser to submit a keystroke even though the system is in X SYSTEM mode.

Set TE_ALLOW_TYPEAHEAD to zero to disable typing ahead, and to any non-zero number to permit typing ahead. You use **TE_setvar** to set the value of TE_ALLOW_TYPEAHEAD. By default, TE_ALLOW_TYPEAHEAD is set to zero, preventing keystrokes from being sent during X SYSTEM mode.

For more information about the **TE_type** function and its conventions, 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 Chapter 64, "Configuring RTE Run-Time Settings."

## Generating Unique Device Names

Some protocols, such as APPC, require a unique device name for each terminal that logs on to the system. Using the run-time settings, you can specify that the **TE_connect** function generate a unique 8-character device name for each Vuser, and connect using this name. Although this solves the requirement for uniqueness, some systems have an additional requirement: The device names must conform to a specific format. For details about the run-time settings, see "Configuring Run-Time Settings" in your VuGen user's guide.

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**.

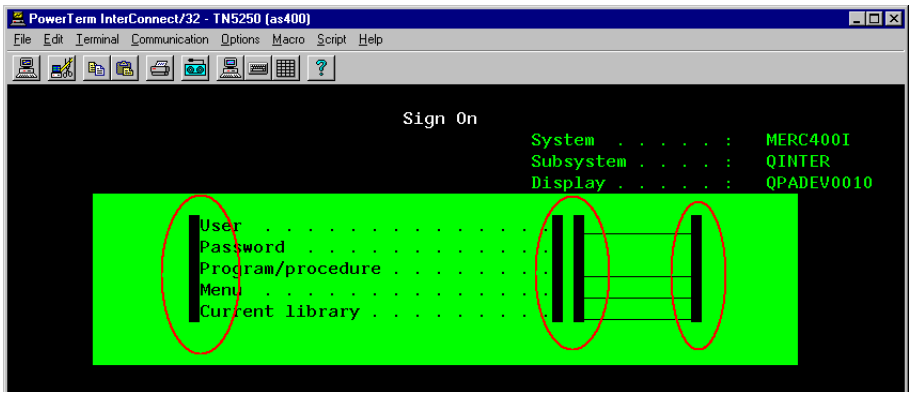
If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

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.

# 64

# 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:

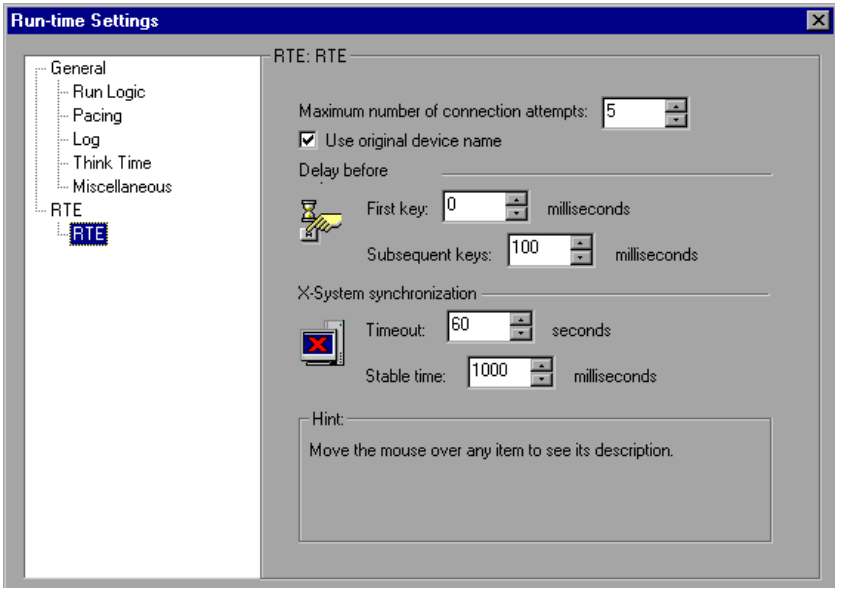
- About Terminal Emulator Run-Time Settings
- Modifying Connection Attempts
- Specifying an Original Device Name
- Setting the Typing Delay
- Configuring the X-System Synchronization

***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 the number of connection attempts, device names, 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 LoadRunner Controller or the Mercury Tuning Module. For more information, refer to the appropriate documentation.

**Note:** 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 12, "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, refer to 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, refer to 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, refer to 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, refer to the *Online Function Reference* (**Help > Function Reference**).

### Timeout

When you replay a **TE_wait_sync** function, if the system does not stabilize before the synchronization timeout expires, the **TE_wait_sync** function returns an error code. To set the synchronization timeout, enter a value (in seconds) in the Timeout section of the RTE Run-Time settings.

The default timeout value is 60 seconds.

### Stable Time

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X-SYSTEM mode. After the terminal returns from the X-SYSTEM mode, the Vuser still monitors the system for a short time. This makes sure that the terminal has become stable, that is, that the system has not returned to the X-SYSTEM mode. Only then does the **TE_wait_sync** function terminate.

To set the time that a Vuser continues to monitor the system after the system has returned from the X-SYSTEM mode, enter a value (in milliseconds) in the Stable time box of the RTE Run-Time settings.

The default stable time is 1000 milliseconds.

# 65

# Synchronizing RTE Vuser Scripts

Synchronization functions in an RTE Vuser script help you synchronize the input that a Vuser submits to a terminal emulator with the responses from the server.

This chapter describes:

- About Synchronizing Vuser Scripts
- 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 1012, 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 verify that the terminal is fully stable, that is, that the system does not return to the X SYSTEM mode. Only then does the **TE_wait_sync** function terminate and allow the Vuser to continue executing its script. The period that the Vuser continues to monitor the system, after the system has returned from the X SYSTEM mode, is known as the stable time. The default stable time is 1000 milliseconds.

You may need to increase the stable time if your system exhibits the following behavior:

When a system returns from the X SYSTEM mode, some systems "flickers" to and from the X SYSTEM for a short period of time until the system stabilizes. If the system remains out of the X SYSTEM mode for more than one second, and then returns to the X SYSTEM mode, the **TE_wait_sync** function will assume that the system is stable. If a Vuser then tries to type information to the system, the system will shift into keyboard-locked mode.

Alternatively, if your system never flickers when it returns from the X SYSTEM mode, you can reduce the stable time to less than the default value of one second.

**To change the stable time for TE_wait_sync functions:**

1. 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("syncTrans1");
```

Each **TE_wait_sync_transaction** function creates a transaction with the name "default." This allows you to analyze how long the terminal emulator waits for responses from the server during a tuning session or scenario run. You use the recording options to specify whether VuGen should generate and insert **TE_wait_sync_transaction** statements.

**To instruct VuGen to insert TE_wait_sync_transaction statements:**

1. 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**).

# 66

# 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:

- About Reading Text from the Terminal Screen
- 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

# 67

# 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:

- About Developing Vuser Scripts for Mailing Services
- 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 5, "Setting Script Generation Preferences." If the communication is performed through multiple protocols, you can record both of them. You can record several mail protocols, or a mail protocol together with HTTP or WinSock. For instructions on specifying multiple protocols, see Chapter 4, "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 4, "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, refer to the *Online Function Reference* (**Help > Function Reference**).

**3 Enhance the script.**

Enhance the script by inserting transactions, rendezvous points, and control-flow structures into the script.

For details, see Chapter 7, "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 8, "Working with VuGen Parameters."

**5 Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 11, "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 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# 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, refer to 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.mycompany.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, refer to 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=Ipm.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, refer to the *Online Function Reference* (**Help** > **Function Reference**).

| Function Name | Description |
|---|---|
| **pop3_command[_ex]** | Sends a command to a POP3 server. |
| **pop3_delete[_ex]** | Deletes a message on the server. |
| **pop3_free[_ex]** | Frees the POP3 server from its commands. |
| **pop3_list[_ex]** | Lists the messages on the POP3 server. |
| **pop3_logoff[_ex]** | Logs off from a POP3 server. |
| **pop3_logon[_ex]** | Logs on to a POP3 server. |
| **pop3_retrieve[_ex]** | 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, refer to the *Online Function Reference* (**Help > Function Reference**).

| Function Name | Description |
|---|---|
| **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=Smtp Test User 0001",
            NULL);

      smtp_send_mail("SendMail",
                        "To=user0002t@merc-int.com",
                        "Subject=MIC Smtp: Sample Test",
                        "MAILOPTIONS",
                        "X-Priority: 3",
                        "X-MSMail-Priority: Medium",
                        "X-Mailer: Microsoft Outlook Express 5.50.400\r\n",
                        "X-MimeOLE: By Microsoft MimeOLE V5.50.00\r\n",
                        "MAILDATA",
                        "MessageText="
                            "Content-Type: text/plain;\r\n"
                            "\tcharset=\"iso-8859-1\"\r\n"
                            "Test,\r\n"
                            "MessageBlob=16384",
                     NULL);

      smtp_logout();

      return 1;
}
```

# Part XIII

## Middleware Protocols

# 68

# 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 Java Vuser API functions.

This chapter describes:

- About Jacada Vuser Scripts
- 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 38, "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 integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. See Chapter 29, "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 29, "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 18, "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 Vuser API functions to the Vuser script. For details, see Chapter 29, "Programming Java Scripts." Use VuGen's Function Navigator to add classes or methods. (See Chapter 55, "Performing EJB Testing.")

**6 Parameterize the Vuser script.**

Replace recorded constants with parameters. You can parameterize complete strings or parts of a string. For details, see Chapter 8, "Working with VuGen 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 20, "Configuring Java Run-Time Settings."

**8 Save and run the Vuser script.**

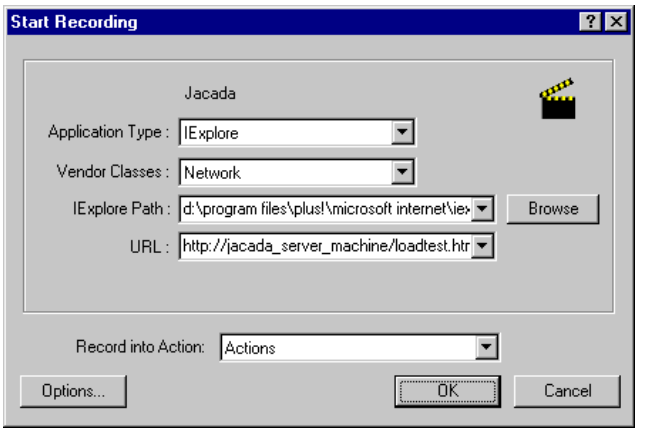
Run the script from VuGen and view the messages in the Execution log tab. For details, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

# 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 of 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, see Chapter 18, “Setting Java Recording Options.”

**6** In the **Record into Action** box, select the section corresponding to the method into which you want to record. The Actions class contains three methods: **init**, **action**, and **end**, corresponding to the vuser_init, Actions, and vuser_end sections. The following table shows what to include into each method, and when each method is executed.

| method within Actions class | Record into action | Used to emulate... | Executed during... |
|---|---|---|---|
| **init** | **vuser_init** | a login to a server | Initialization |
| **action** | **Actions** | client activity | Running |
| **end** | **vuser_end** | a log off procedure | Finish or Stopped |

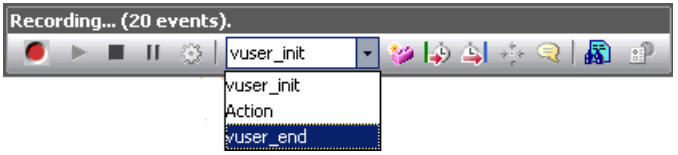
**7** Click **OK** to begin recording. VuGen starts your application, minimizes itself and opens a progress bar and the floating recording toolbar. The progress toolbar displays the names of classes as they load. This indicates that the Java recording support is active.




**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 the Vusers—JRE alone is insufficient.Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions. Before you replay a Vuser script, verify that your environment is configured properly for the JDK and relevant Java classes.

Before replay, you must also download the **clbase.jar** file from the Jacada server. All classes used by the Java Vuser must be in the classpath—either set in the machine's CLASSPATH environment variable or in the **Classpath Entries** list in the Classpath node of the Run-Time settings.

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 29, "Programming Java Scripts."

# Understanding Jacada Vuser Scripts

When you record a Jacada session, VuGen logs all calls to the server and generates a script. 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 Vuser API 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, refer to 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 Java Vuser API functions. For a list of these functions and information on how to add them to your script, see Chapter 29, "Programming Java Scripts."

# 69

# 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:

- About Tuxedo Vuser Scripts
- 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, 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 4, "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 7, "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 8, "Working with VuGen Parameters."

**4 Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 11, "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 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## 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

| | |
|---|---|
| **lrt_Fadd[32]_fld** | Adds a new field to an FML buffer. |
| **lrt_Finitialize[32]** | Initializes an existing FML buffer fbfr. |
| **lrt_Fldid[32]** | Maps a field name to a field identifier. |
| **lrt_Fname[32]** | Provides a map field identifier to field name. |
| **lrt_memcpy** | Copies the specified amount of bytes from the source to the destination. |
| **lrt_strcpy** | Copies a string like the C function strcpy. |
| **lrt_tpalloc** | Returns a pointer to a buffer type of *type*. |
| **lrt_tprealloc** | Changes the size of a typed buffer. |
| **lrt_tpfree** | Frees a typed buffer. |
| **lrt_tptypes** | Determines information about a typed buffer. |

### Client/Server Session Functions

| | |
|---|---|
| **lrt_tpchkauth** | Checks if authentication is required by the application. |
| **lrt_tpinitialize** | Enables a client to join a System/T application. |
| **lrt_tpterm** | Removes a client from a System/T application. |

### Communication Functions

| | |
|---|---|
| **lrt_tpacall** | Sends a service request. |
| **lrt_tpbroadcast** | Broadcasts notification by name. |
| **lrt_tpcall** | Sends a service request and awaits its reply. |
| **lrt_tpcancel** | Cancels a call descriptor. |
| **lrt_tpchkunsol** | Checks for an unsolicited message. |
| **lrt_tpconnect** | Establishes a conversational service connection. |
| **lrt_tpdequeue** | Dequeues a message from a queue. |
| **lrt_tpdiscon** | Terminates a conversational service connection. |
| **lrt_tpenqueue** | Stores a message in the queue. |
| **lrt_tpgetrply** | Returns a reply from a previously sent request. |
| **lrt_tpgprio** | Returns the priority for the last request sent or received. |
| **lrt_tpnotify** | Sends notification to a client. |
| **lrt_tprecv** | Receives a message in a conversational connection. |
| **lrt_tpsend** | Sends a message in a conversational connection. |

| | |
|---|---|
| **lrt_tpsetunsol** | Sets the method for handling unsolicited messages. |
| **lrt_tpsprio** | Sets the priority for the next request sent or forwarded. |
| **lrt_tpsubscribe** | Subscribes to an event. |
| **lrt_tpunsubscribe** | Unsubscribes to an event. |

### Environment Variable Functions

| | |
|---|---|
| **lrt_set_env_list** | Sets a list of environment variables. |
| **lrt_tuxgetenv** | Returns a value corresponding to an environment name. |
| **lrt_tuxputenv** | Modifies an existing environment value or adds a value to the environment. |
| **lrt_tuxreadenv** | Adds variables to the environment from a file. |

### Error Processing Functions

| | |
|---|---|
| **lrt_abort_on_error** | Aborts the current transaction, if the previous Tuxedo function call resulted in an error. |
| **lrt_Fstrerror[32]** | Retrieves error message string for FML error. |
| **lrt_getFerror[32]** | Retrieves the error status code for the last FML operation that failed. |
| **lrt_gettperrno** | Retrieves the error status code for the last Tuxedo transaction monitor function. |
| **lrt_gettpurcode** | Retrieves the application return code. |
| **lrt_tpstrerror** | Retrieves error message string for System/T error. |

### Transaction Handling Functions

| | |
|---|---|
| **lrt_tpabort** | Aborts the current transaction. |
| **lrt_tpbegin** | Begins a transaction. |
| **lrt_tpcommit** | Commits the current transaction. |
| **lrt_tpgetlev** | Checks if a transaction is in progress. |
| **lrt_tpresume** | Resumes a global transaction. |
| **lrt_tpscmt** | Sets when lrt_tpcommit should return. |
| **lrt_tpsuspend** | Suspends a global transaction. |
| **lrt_tx_begin** | Begins a global transaction. |
| **lrt_tx_close** | Closes a set of resource managers. |
| **lrt_tx_commit** | 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 commit_return characteristic to the value specified in when_return. |
| **lrt_tx_set_transaction _control** | Sets the **transaction_control** characteristic to the value specified in control. |
| **lrt_tx_set_transaction _timeout** | Sets the transaction_timeout characteristic to the value specified in timeout. |

**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 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. |

---

**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);
tpresult_int = lrt_tpbegin(30, 0);
data_0 = lrt_tpalloc("FML", "", 512);
lrt_Finitialize((FBFR*)data_0);

/* Fill the data buffer data_0 with new account information */
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=8",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=C",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=MID_INIT", "value=Q",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value=123-456-7890",
        LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
        New York, NY 10000", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=SSN","value=111111111",
LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0,"name=LAST_NAME",
"value=Doe",LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0,"name=FIRST_NAME",
"value=BJ",LRT_END_OF_PARMS);

lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT",
"value=0.00",LRT_END_OF_PARMS);

/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, &data_0, &olen_2, 0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();
```

### Using Parameters in Tuxedo Scripts

You can define parameters in Tuxedo scripts, as described in Chapter 8, "Working with VuGen Parameters." Note that Tuxedo scripts contain strings of type "name=..." or "value=...". You can only define parameters for the portion of the string following the equal sign (=). For example:

```
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value=<parameter_1>",
                LRT_END_OF_PARMS);
```

### Running Tuxedo Scripts

If you encounter problems recording or running Tuxedo applications, check that the Tuxedo application runs without VuGen, and that the environment variables have been defined correctly. For more information, see "Viewing Tuxedo Buffer Data" below. Note that after you set or modify the Tuxedo variables, you should restart VuGen and your application, in order for the changes to take effect. If your application is 16-bit, then you also need to kill the NTVDM process.

If you experience problems during execution, check the Tuxedo log file on the side of the server for error messages. By default, this file is found in the directory indicated by the environment variable APPDIR. The file name has the form ULOG.mmddyy, where mmddyy indicates the current month, day, and year. The file for March 12, 1999 would be ULOG.031299. The default location of this file can be changed by setting the environment variable ULOGPFX on the server. A log file can also be found on the client side, in the current directory, unless the ULOGPFX variable changes its location.

## Viewing Tuxedo Buffer Data

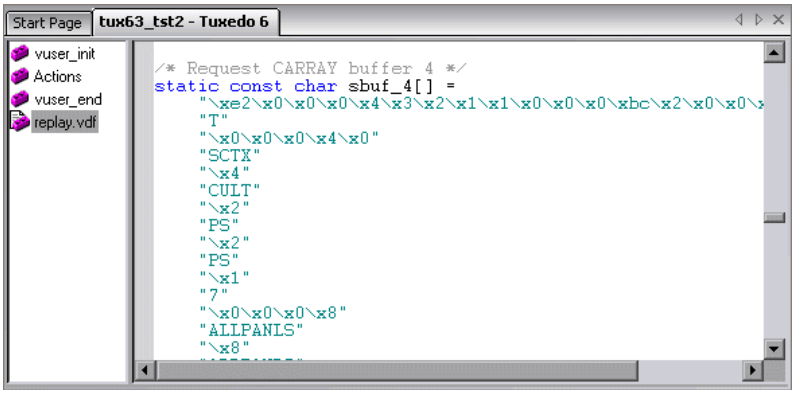
When you use VuGen to create a Tuxedo Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**.

The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file is called **replay.vdf**, and it contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRT functions use the buffer descriptors to access the data.

You can use VuGen to view the contents of the data file by selecting the **replay.vdf** file in the left pane's tree view.

The option to view a data file is available by default for Tuxedo scripts.



# Defining Environment Settings for Tuxedo Vusers

The following section describes the system variable settings for Tuxedo Vusers running on Windows and UNIX platforms. You define the system variables in your Control Panel/System dialog box (NT) or .cshrc or .login file (UNIX).

| | |
|---|---|
| TUXDIR | the root directory for Tuxedo sources. |
| FLDTBLDIR | list of directories containing FML buffer information. In Windows, separate the names of directories with semi-colons. On UNIX platforms, separate the names of the directories with a colon. |
| FIELDTBLS | list of files containing FML buffer information. On both Windows and UNIX platforms, separate the file names with commas. |

For example:

```
SET FLDTBLDIR=%TUXDIR%\udataobj;%TUXDIR%\APPS\WS (PC)
SET FIELDTBLS=bankflds,usysflds (PC)
setenv FLDTBLDIR $TUXDIR/udataobj:$TUXDIR/apps/bankapp (Unix)
setenv FIELDTBLS bank.flds,Usysflds (Unix)
```

You must define the following system variables for Tuxedo clients using Tuxedo/WS workstation extensions during execution:

| | |
|---|---|
| WSNADDR | specifies the network address of the workstation listener process. This enables the client application to access Tuxedo. Note that to define multiple addresses in a WSNADDR statement, each address must be separated by a comma. |
| WSDEVICE | specifies the device that accesses the network. Note that you do not need to define this variable for some network protocols. |

For example:

```
SET WSNADDR=0x0002ffffc7cb4e4a (PC)
setenv WSNADDR 0x0002ffffc7cb4e4a (Unix)
setenv WSDEVICE /dev/tcp (Unix)
```

## Debugging Tuxedo Applications

In general, use **Tuxedo 6** to record applications using Tuxedo 6.x or earlier, and use **Tuxedo 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_tpalloc("FML", "", 512);
lrt_Finitialize((FBFR*)data_0);
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=1",
LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=S",
LRT_END_OF_PARMS);
…

LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=LAST_NAME", "value=Doe", …);
lrt_Fadd_fld((FBFR*)data_0, "name=FIRST_NAME", "value=John", …);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=234.12", …);

/* Open a new account and save the new account number*/
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0,&data_0, &olen_2, 0);
lrt_abort_on_error();
lrt_save_fld_val((FBFR*)data_0, "name=ACCOUNT_ID", 0,
"account_id");
```

```
/* Use result from first query to fill buffer for the deposit*/
lrt_Finitialize((FBFR*)data_0);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCOUNT_ID",
"value=<account_id>", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=200.11", …);
```

In the above example, the account ID was represented by a field name, ACCOUNT_ID. Some systems represent a field by an ID number rather than a field name during recording.

You can correlate by field ID as follows:

```
lrt_save_fld_val((FBFR*)data_0, "id=8302", 0, "account_id");
```

### Correlating Character Strings

Use **lrt_save_parm** or **lrt_save_searched_string** to correlate character strings.

- In general, 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

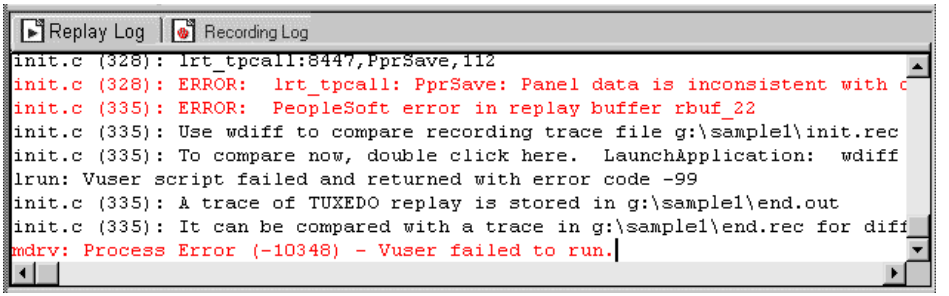
When working with CARRAY buffers, VuGen generates log files during recording (with the **.rec** extension) and during replay (with the **.out** extension) which you can compare using the Wdiff utility. You can look at the differences between the recording and replay logs to determine which portions of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

The error message should be followed by a statement beginning with the phrase: **Use wdiff to compare**.




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

WDiff opens and the differences between the record and replay files are highlighted in yellow. For more details about the Wdiff utility, see Chapter 11, "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);
      tpresult_int = lrt_tpcall("PprLoad",
           data_0, 227, &data_1, &olen, TPSIGRSTRT);
      lrt_save_parm(data_1, 123, 9, "empid");
```

In the **replay.vdf** file, replace the recorded value, "G001", with the parameter, **empid**.

```
char buf_143[] =
"\xf5\x0\x0\x0\x4\x3\x2\x1\x1\x0\x0\x0\xbc\x2\x0\x0\x0\x0\x0\x0"
   "X"
"\x89\x0\x0\x0\xb\x0"
   "SPprLoadReq"
   "\xff\x0\x10\x0\x0\x4\x3\x6"
   "<empid>" // G001
   "\x7"
   "Claudia"
   "\xe"
   "LAST_NAME_SRCH"
...
```

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

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

**To correlate statements using lrt_save_searched_string:**

Use **lrt_save_searched_string** to search for a string in a buffer, and save a portion of the buffer, relative to the string occurrence, to a parameter.

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

```
lrt_save_searched_string (buffer, buf_size, occurrence, string, offset,
                                    length, "param_name");
```

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

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

View the buffer contents by selecting the **replay.vdf** file in the Data Files box of the main VuGen window.

**3** Replace all instances of the value with the parameter name in 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 come 9 bytes past the first occurrence of "SCertRep", but the length of the information before this string varies.

```
/* Request CARRAY buffer 1 */
lrt_memcpy(data_0, sbuf_1, 41);
lrt_display_buffer("sbuf_1", data_0, 41, 41);
data_1 = lrt_tpalloc("CARRAY", "", 8192);
tpresult_int = lrt_tpcall("GetCertificate",
      data_0,
      41,
      &data_1,
      &olen,
      TPSIGRSTRT);

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

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

# Part XIV

## Streaming Data Protocols

# 70

# 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 is an application that display streaming content.

You use VuGen to record communication between a client application and a server that communicate using the RealPlayer protocol. The resulting script is called a Real Vuser script.

This chapter describes:

- About Recording Streaming Data Virtual User Scripts
- Getting Started with Streaming Data Vuser Scripts
- Using RealPlayer LREAL Functions
- Using Media Player MMS Functions

***The following information applies only to the Real and Media Player (MMS) 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 Real or Media Player Vuser script:**

1 **Record the basic script using VuGen. (Real only)**

Invoke VuGen and create a new Vuser script. Choose an application to record, and record typical operations on your application. For details, see Chapter 4, “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 7, “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 8, “Working with VuGen Parameters.”

4 **Correlate statements (optional).**

Correlating statements enables you to use the result of one business process in a subsequent one.

For details, see Chapter 11, “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 12, "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 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# Using RealPlayer LREAL Functions

The functions developed to emulate communication between a client and a server by using the RealPlayer protocol are called Real Player functions. Each Real Player function has an **lreal** prefix. VuGen automatically records most of the LREAL functions listed in this section during a Real Player session. You can also manually program any of the functions into your script. For 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, refer to 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

# 71

# 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:

- About Wireless Vusers
- Understanding the WAP Protocol
- Understanding the i-mode System
- i-mode versus WAP
- Understanding VoiceXML

## About Wireless Vusers

VuGen supports three 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.

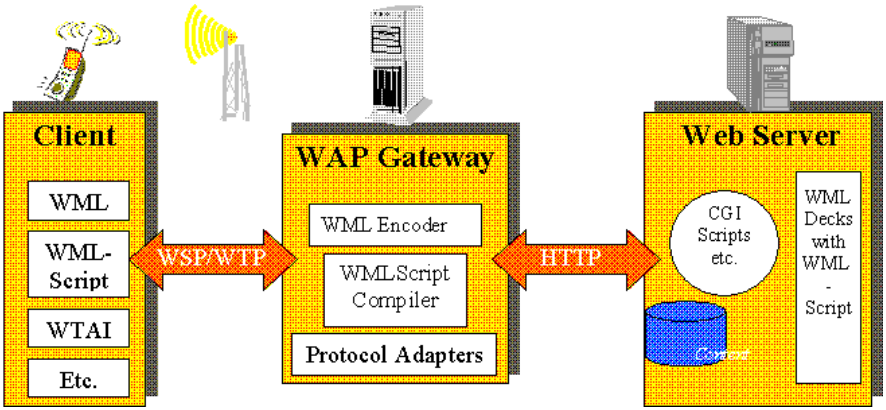
WAP also specifies a proxy server that:

- 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 1086.

## 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 Pixo 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-vxml, */*");

	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;
}
```

# 72

# 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:

- About Recording Wireless Vuser Scripts
- 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 scenario or 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 4, "Recording with VuGen."

**2 Set the recording options.**

Set the recording options. For information about setting common Internet recording options, see Chapter 40, "Setting Recording Options for Internet Protocols." For information about Wireless specific recording options, see Chapter 74, "Setting Wireless Vuser Recording Options."

**3 Record the actions using VuGen.**

Record the actions over the toolkit session.

For information about recording, see Chapter 4, "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 7, "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 8, "Working with VuGen 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 12, "Configuring Run-Time Settings."

For information about common Internet protocol run-time settings, see Chapter 42, "Configuring Internet Run-Time Settings."

For information about WAP specific run-time settings, see Chapter 75, "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 687.

**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 standalone test, see Chapter 14, "Running Vuser Scripts in Standalone Mode."

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

## Using Wireless Vuser Functions

The functions developed to emulate communication between a wireless instrument and Web server (or gateway for WAP), are called Vuser functions. Some functions are generated when you record a script; others you must manually insert into the script. You can also add Vuser message functions and custom C functions to your Vuser scripts after recording.

The functions representing standard HTTP actions, have a **web** prefix. For information about these functions, see Chapter 39, "Using Web Vuser Functions."

General Vuser functions begin with an **lr** prefix. For more information, see "Using C Vuser Functions" on page 32.

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 39, "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** |

### WAP Specific Functions

| | |
|---|---|
| **wap_add_const_header** | Specifies a constant header to pass to a WAP gateway. |
| **wap_connect** | Connects to a WAP gateway. |
| **wap_disconnect** | Disconnects from a WAP gateway. |
| **wap_format_si_message** | Formats an SI type message |
| **wap_format_sl_message** | Formats an SL type message |
| **wap_pi_push_cancel** | Cancels a message sent to a PPG. |
| **wap_pi_push_submit** | Submits a Push message. |
| **wap_radius_connection** | Connects or disconnects from a RADIUS server. |
| **wap_send_sms** | Sends an SMS type message. |
| **wap_set_bearer** | Sets the underlying bearer-UDP or CIMD2 (SMS). |
| **wap_set_capability** | Sets a client capability for a WAP gateway connection. |
| **wap_set_connection_mode** | Sets the connection mode and security level. |
| **wap_set_gateway** | Sets a gateway IP address and port. |
| **wap_set_sms_user** | Sets login information for the SMSC. |
| **wap_wait_for_push** | 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 (3.0 and 3.1), 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. (You must invoke the toolkit after starting VuGen.)
4. In the Recording Options, choose the **Internet Protocol:WAP Toolkit** node and select **Manually launch a WAP toolkit**.
5. Click **OK**. A message box opens with the Gateway IP address assigned by VuGen.
6. Copy the IP address and paste it 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.

---

# 73

# 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:

- About WAP Vusers
- Recording Over a Phone
- Bearers Support
- RADIUS Support
- Push Support
- VuGen Push 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 71, "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 1086.

You can record a WAP session using a toolkit or through your phone. For information about recording through a toolkit, see Chapter 74, "Setting Wireless Vuser Recording Options." For information about recording over a phone, see "Recording Over a Phone" below.

You can program scripts to emulate WAP sessions using the **wap** Vuser functions. For more information and examples, refer to 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 74, "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 Radius Connection Data" on page 1111).

➤ **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 75, "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 75, "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 basic proxy gateway services, the PPG is capable of notifying the Push Initiator about the final status of the push operation. In two-way mobile networks, it can also wait for the client to accept or reject the content.

### Push Services Types

Push services can be of the SL or SI type:

- **SL** - The Service Loading (SL) content type provides the ability to cause a user agent on a mobile client to load and execute a service—for example, a WML deck. The SL contains a URI indicating the service to be loaded by the user agent without user intervention when appropriate.
- **SI** - The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. For example, the notifications may be about new 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.

## VuGen Push Support

Push support for VuGen is divided into three parts:

- Push support at the client end—the ability to accept push messages.
- Push support to WAP HTTP Vusers—emulating Push Initiators.
- Push messages (SI & SL) format services—formatting push messages.

### Client Push Support

At the client end, VuGen supports both push services (SL and SI) for all replay modes (CO and CL). The **wap_wait_for_push** function instructs the Vuser to wait for a push message to arrive. You set the timeout for this function in the run-time settings.

When a push message arrives, the Vuser parses it to determine its type and to retrieve its attributes. If parsing was successful, it generates and executes a pull transaction to retrieve the relevant data. You can disable the pull event, indicating to the Vuser not to retrieve the message data by configuring the Run-Time settings. For more information, see Chapter 75, "Configuring WAP Run-Time Settings."

### Emulating a Push Initiator

Push support for WAP HTTP Vusers enables you to perform load testing of the PPG. Push support allows Vusers to function as Push Initiators supporting the **Push Access Protocol** (PAP). The PAP defines the following sets of operations between the PI and the PPG:

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 VuGen. Currently, only the first two operations are supported through **wap_push_submit** and **wap_push_cancel.**

   You can submit data to a Web server using the **web_submit_data** function. It is difficult, however, to send long and complex data structures using this function. To overcome this difficulty and provide a more intuitive API function, several new API functions were added to properly format the XML message data: **wap_format_si_msg** and **wap_format_sl_msg**. For more information about these functions, refer to the *Online Function Reference*.

# 74

# Setting Wireless Vuser Recording Options

Before recording a Wireless session, you can customize the recording options.

This chapter describes:

- About Setting Wireless Recording Options
- Specifying the Recording Mode (WAP only)
- Specifying the Information to Record (i-mode and VoiceXML)
- Specifying a Toolkit

## About Setting Wireless Recording Options

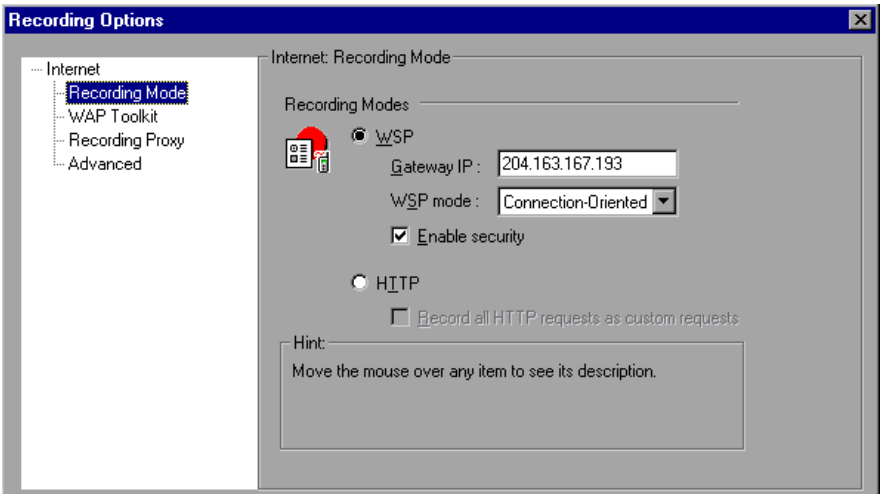
VuGen enables you to generate Wireless Vuser scripts by recording typical processes that users perform on your Web site using their wireless interfaces.

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 40, "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.

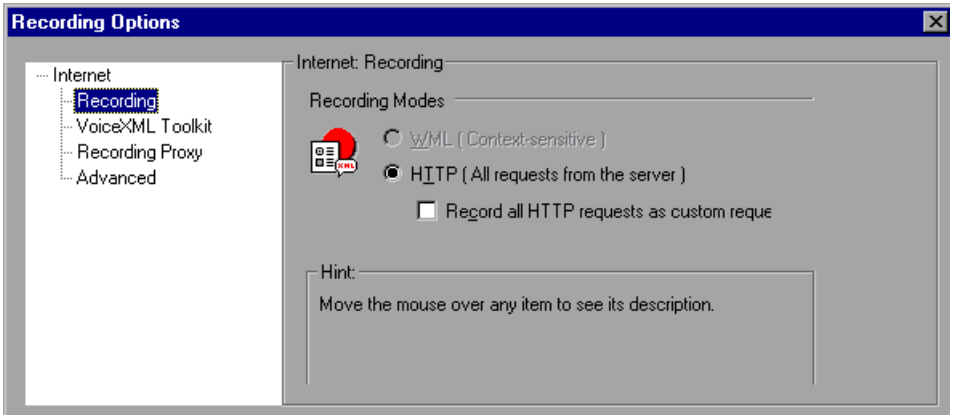
For recording in WSP mode, VuGen contains native support for the Phone.com UP Simulator 4.1 toolkit. It detects the installation, automatically sets the configuration parameters, and launches it. For Nokia toolkits (1.3 and 2.0), you need to launch the toolkit manually assigning it the proper IP address. For more information, see “Troubleshooting Wireless Vuser Scripts” on page 1087.

➤ **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. You can select both an i-mode Recording Mode and a VoiceXML Recording Mode.

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 or VoiceXML 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.

### i-mode Recording Mode

This node allows you to specify a recording mode for i-mode Vusers.

The recording mode that you select depends on your needs and environment. The available recording modes are HTTP or Custom Requests.

**HTTP:** Captures all HTTP requests sent to the server as a result of user actions, creating **web_url** statements for each request. This recording level captures even non-HTML applications such as applets and non-browser applications.

**Record All HTTP Requests as Custom Requests:** Records all HTTP requests as custom HTTP requests, disregarding their contexts. Custom requests do not rely on a specific structure or the HTTP request statement. VuGen generates a **web_custom_request** function for each page and resource that it records, instead of the **web_url**, **web_image**, or **web_submit_form** functions.

### VoiceXML Recording Mode

This node allows you to specify a recording mode for VoiceXML Vusers.

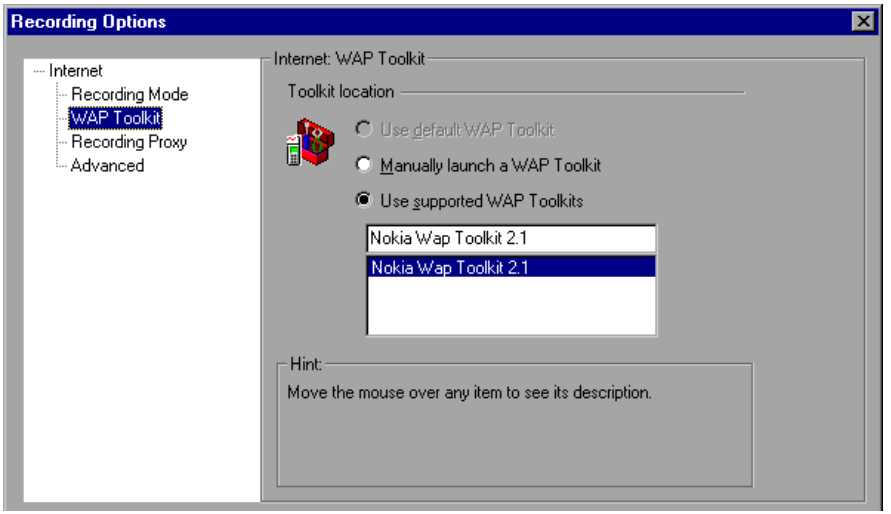
The recording mode that you select depends on your needs and environment. The available recording modes are HTTP or Custom Requests.

**HTTP:** Captures all HTTP requests sent to the server as a result of user actions, creating **web_url** statements for each request. This recording level captures even non-HTML applications such as applets and non-browser applications.

**Record All HTTP Requests as Custom Requests:** Records all HTTP requests as custom HTTP requests, disregarding their contexts. Custom requests do not rely on a specific structure or the HTTP request statement. VuGen generates a **web_custom_request** function for each page and resource that it records, instead of the **web_url**, **web_image**, or **web_submit_form** 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 VoiceXML Toolkit, i-mode Toolkit, or VoiceXML Toolkit.




The following options are available:

- **Use default... toolkit** - record using the default toolkit.
- **Manually launch... toolkit** - manually launch a toolkit.
- **Use supported... toolkits** - Use a toolkit from the list of supported toolkits.

### WAP Toolkit

In this node, you indicate which WAP toolkit to use during recording. The supported toolkits that are installed are listed below.

**To specify the toolkit for recording a WAP 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.

### i-mode Toolkit

In this node, you indicate which i-mode toolkit to use during recording. The supported toolkits that are installed are listed below.

**To specify the toolkit for recording an i-mode Vuser script:**

1 Choose **Tools** > **Recording Options** and select the **i-mode Toolkit** node.

2 Select one of the following options in the **Toolkit Location** section:

- **Use default i-mode Toolkit:** Instructs VuGen to use the default toolkit on the recording computer (currently disabled).
- **Manually launch a i-mode 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 i-mode Toolkits:** Instructs VuGen to use a specific toolkit installed on the machine. Select one of the available toolkits listed in the dialog box.

## VoiceXML Toolkit

In this node, you indicate which VoiceXML toolkit to use during recording. The supported toolkits that are installed are listed.

**To specify the toolkit for recording a VoiceXML Vuser script:**

1 Choose **Tools** > **Recording Options** and select the **VoiceXML Toolkit** node.

2 Select one of the following options in the **Toolkit Location** section:

- **Use default VoiceXML Toolkit:** Instructs VuGen to use the default toolkit on the recording computer (currently disabled).
- **Manually launch a 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 VoiceXML Toolkit:** Instructs VuGen to use a specific toolkit installed on the machine. Select one of the available toolkits listed in the dialog box.

# 75

# Configuring WAP Run-Time Settings

After you record a WAP Vuser script, you configure the WAP specific run-time settings.

This chapter describes:

- About WAP Run-Time Settings
- Configuring Gateway Options
- Configuring Radius Connection Data

For information on setting the common Internet protocol run-time settings for WAP and all Wireless protocols, see Chapter 42, "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 and Radius 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.

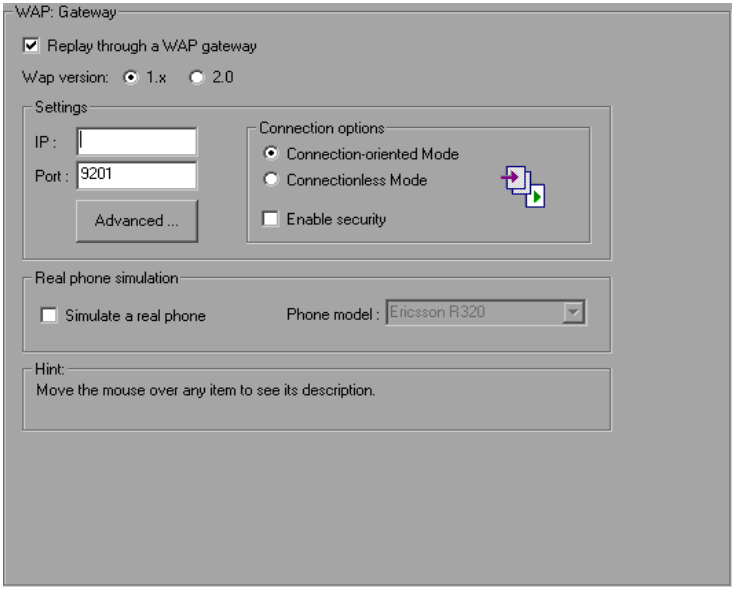


To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar.

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 42, "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.




### Communication Protocol

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.

**Wap version**: Select the appropriate WAP version, 1.x or 2.0.

### Settings

**IP:** Specify the IP address of the gateway.

**Port:** Specify the port of the gateway. When running your Vusers through a WAP gateway, VuGen automatically sets default port numbers, depending on the selected mode. However, you can customize the settings and specify a custom IP address and port for the gateway.

**Advanced:** Opens the Advanced Gateway Options dialog box, where you can set the client capabilities and other advanced gateway options.

### Connection Settings

In this section you indicate the desired replay connection mode.

- **Connection-oriented Mode:** Sets the connection mode for the WSP session to Connection-Oriented.
- **Connectionless Mode:** Sets the connection mode for the WSP session to Connectionless.
- **Enable security:** Enables a secure connection to the WAP gateway.

### 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.

- **Simulate a real phone:** Instructs VuGen to simulate a real phone.
- **Phone model:** Select the phone model to simulate from this menu.

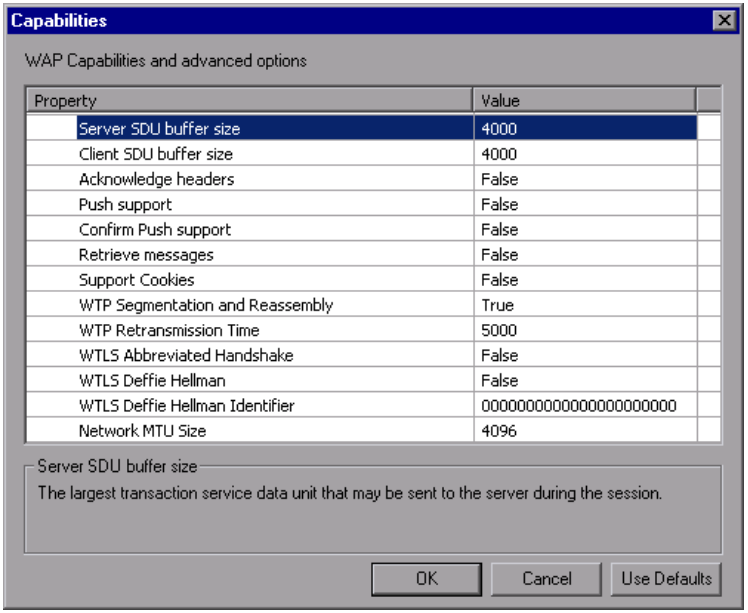
Note that when you enable real phone simulation, all of the Advanced gateway options are ignored. Instead it retrieves the header and client capability information from the VuGen configuration file which defines each of the supported phones.

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, **LrwWapPhoneDB.dat** in your application's **dat** directory. For more information, see the comments at the beginning of the configuration file.

## Advanced Gateway Options

When you click **Advanced** in the Gateway node, VuGen opens the Capabilities dialog box. In this dialog box you configure the WAP Capabilities and other advanced gateway options.



- **Server SDU buffer size** - the largest transaction service data unit that may be sent to the server during the session (4000 by default).
- **Client SDU buffer size** - the largest transaction service data unit that may be sent to the client during the session (4000 by default).
- **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).
- **WTP Segmentation and Reassembly**- Enables segmentation and reassembly (SAR) in WTP, Wireless Transport Protocol. (True by default).
- **WTP Retransmission Time**- The time in seconds that the WTP layer waits before resending the PDU if it did not receive a response. (5000 by default).
- **WTLS Abbreviated Handshake**- Use an abbreviated handshake instead of a full one, when receiving a redirect message. (False by default).
- **WTLS Deffie Hellman**- Use the Deffie Hellman encryption scheme for WTLS (Wireless Transport Layer Security) instead of the default scheme, RSA. (False by default).
- **WTLS Deffie Hellman identifier**- An identifier for the Deffie Hellman encryption scheme. This identifier is required for the abbreviated handshake with the Operwave gateway that uses the Deffie Hellman encryption scheme.
- **Network MTU Size**- the maximum size in bytes, of the network packet. (4096 by default).

### Setting the Gateway Options

The following section describes the procedure for setting the WAP Gateway options.

**To set the WAP gateway options:**



1. Click the **Run-Time Settings** button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **WAP:Gateway** node.
2. To replay the script in WSP mode (not HTTP), select **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 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.

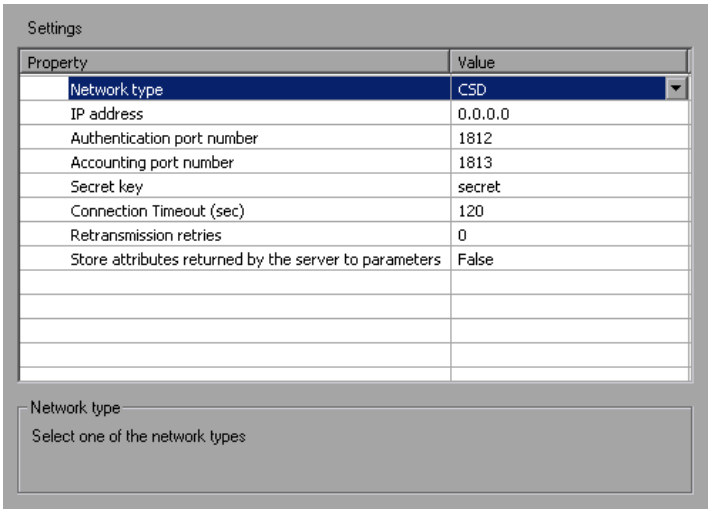
You supply the dial-in information in the Run-Time Settings' **Radius** node:

| Property | Value |
|---|---|
| **Network Type** | Accounting network type: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data). |
| **IP Address** | IP address of the Radius server. |
| **Authentication port number** | Authentication port of the Radius server. |
| **Accounting port number** | Accounting port of the Radius server. |
| **Secret Key** | The secret key of the Radius server. |
| **Connection Timeout (sec)** | The time in seconds to wait for the Radius server to respond. The default is 120 seconds. |
| **Retransmission retries** | The number of times to retry after a failed transmission. The default is 0. |
| **Store attributes returned by the server to parameters** | Allow Vusers to save attributes returned by the server as parameters, which can be used at a later time. The default is **False**. |

**To set the WAP Radius options:**



**1** Click the **Run-Time Settings** button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Click the **Radius** node.




**2** Choose an accounting **Network type**: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).

**3** Enter the **IP address** of the Radius server in dot form.

**4** Enter the **Authentication Port number** and **Accounting Port number** of the Radius server.

**5** Type in the **Secret key** for Radius or Accounting Authentication.

**6** Enter a **Connection Timeout** value.

**7** Specify the number of **Retransmission retries**.

**8** Specify whether you want VuGen to **store attributes returned by the server to parameters**.

**9** Click **OK** to accept the settings and close the dialog box.

# Part XVI

## Information for Advanced Users

# 76

# 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:

- About Creating Vuser Scripts in Visual Studio
- 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.

| | |
|---|---|
| *VuGen* | You can use VuGen to create Vuser script that run on Windows or UNIX platforms by recording or by manually programming within the VuGen editor. You create the script in a Windows environment and run it in either Windows or UNIX—recording is not supported on UNIX. |
| *Visual Studio* | For users working with Visual Studio, you can program in Visual Basic, C or C++. The programs must be compiled into a dynamic link library (dll). |

This chapter describes how to develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API libraries.

You can also program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes. Programming within VuGen is available for C, Java, Visual Basic, VBScript, and JavaScript. For more information, see Chapter 28, "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
- Vuser API calls
- source code and makefiles for creating a dynamic library

After creating a basic Vuser script from a template, you can enhance the script to provide run-time information and statistics. For more information, see Chapter 7, "Enhancing Vuser Scripts."

An online C reference of the common functions used in Vuser scripts, are included in the *Online Function Reference* (**Help** > **Function Reference**).

## Creating a Vuser Script with Visual C

Please note that you can create Vuser scripts using Visual C version 6.0 or higher.

**To create a Vuser script with Visual C:**

1. In Visual C, create a new project - dynamic link library (dll). Choose **File** > **New** and click the Projects tab.
2. In the Wizard, choose *empty dll*.
3. Add the following files to the project:
   - A new *cpp* file with 3 exported function: *init*, *run*, *end* (the names may be customized).
   - The library file lrun50.lib (located in the <lr installation dir>/lib).

4 In the project settings change the following:

- Select the C/C++ tab and choose **Code generation** (Category) > **Use Run Time library** (List). Change it to: **Multithreaded dll.**
- Select the C/C++ tab and choose **Preprocessor** (Category) > **Preprocessor definitions** (edit field) Remove _DEBUG.

5 Add code from your client application, or program as you normally would.

6 Enhance your script with Vuser API functions. For example, **lr_output_message** to issue messages, **lr_start_transaction** to mark transactions, and so forth. For more information, refer to the General functions in the *Online Function Reference* (**Help > Function Reference**).

7 Build the project. The output will be a DLL.

8 Create a directory with the same name as the DLL and copy the DLL to this directory.

9 In the **lrvuser.usr** file in the *Template* directory, Update the USR file key *BinVuser* with the DLL name: BinVuser=<*DLL_name*>.

In the following example, the **lr_output_messsage** function issues messages indicating which section is being executed. The **lr_eval_string** function retrieves the name of the user. To use the following sample, verify that the path to the Vuser API include file, *lrun.h* is correct.

```
#include "c:\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 **LoadRunner Virtual User**. A new project is created with one class and a template for a Vuser.
3. Save the project before you continue to program. Chose **File** > **Save Project**.

4 Open the Object Browser (**View** menu). Select the *LoadRunner Vuser* library and double-click on the Vuser Class module to open the template. The template contains three sections, Vuser_Init, Vuser_Run, and Vuser_End.

```
Option Explicit

Implements Vuser

Private Sub Vuser_Init()
'Implement the Vuser initialization code here
End Sub

Private Sub Vuser_Run()
'Implement the Vuser main Action code here
End Sub

Private Sub Vuser_End()
'Implement the Vuser termination code here
End Sub
```

5 Add code from your client application, or program as you normally would.

6 Use the Object Browser to add the desired VuGen elements to your code, such as transactions, think time, rendezvous, and messages, using the object browser.

7 Enhance your program with run-time settings and parameters. For more information, see "Configuring Runtime Settings and Parameters" on page 1120.

8 Build the Vuser script: select **File** > **Make** *project_name*.dll.

The project is saved in the form of a Vuser script (.usr). The script resides in the same directory as the project.

# Configuring Runtime Settings and Parameters

After you create the DLL for your script, you create a script (*.usr*) and configure its settings. The *lrbin.bat* utility provided with VuGen lets you define parameters and configure runtime settings for scripts created with Visual C and Basic. This utility is located in the *bin* directory of the product installation.

**To configure runtime settings and parameterize scripts:**

1 In the product's *bin* directory, double-click on *lrbin.bat*. The Standalone Vuser Configuration dialog box opens.




2 Choose **File** > **New**. Specify a script name for the *usr* file. The script name must be identical to the name of the directory to which you saved the DLL.

3 Choose **Vuser** > **Advanced** and enter the DLL name in the Advanced dialog box.

4 Choose **Vuser** > **Run-time Settings** to define run-time settings. The Run-time Settings dialog box is identical to that displayed in the VuGen interface. For more information, see Chapter 12, "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 8, "Working with VuGen Parameters."

Test the script by running it in standalone mode. Choose **Vuser** > **Run Vuser**. The Vuser execution window appears while the script runs.

6 Choose **File** > **Exit** to close the configuration utility.

# 77

# 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:

- About Programming with the XML API
- 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, Web Services, and Wireless Vuser scripts.***

## About Programming with the XML API

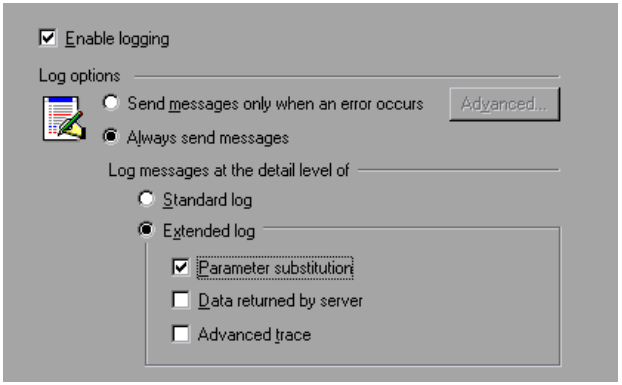
VuGen's support for XML allows you to dynamically work with XML code and retrieve the values during test execution. Follow these steps in creating an effective XML script:

- Record a script in the desired protocol, usually Web, Web Services, or Wireless.
- Copy the XML structures into your script.

➤ Add XML functions from the LR API in order to retrieve dynamic data and the XML element values.

The LR API uses XPath, the XML Path language to manipulate the text in an XML document.

You can instruct VuGen to display the output values of XML elements in the Execution log window using the Run-Time settings. VuGen displays the line numbers, the number of matches, and the value. To allow the displaying of values, you need to enable parameter substitution. In the Run-Time settings, open the **General:Log** node, select **Extended log**, and choose **Parameter Substitution**. For more information, see Chapter 12, "Configuring Run-Time Settings."




All Vuser API XML functions return the number of matches successfully found, or zero for failure.

# Understanding XML Documents

XML, or Extensible Markup Language, is a markup language that you can use to create your own custom tags. Using these tags, you give a meaning to the text between the tags. This stands in contrast to standard HTML tags such as H1, P, DIV, and so on, which cannot be customized and do not indicate the content of the text.

XML documents consist of trees with many nodes and branches. There are three common terms used that describe the parts of an XML document: *tags*, *elements*, and *attributes*. The following example illustrates these terms:

```
<acme_org>
    <accounts_dept>
        <employee type='PT'>
            <name>John Smith</name>
            <cubicle>227</cubicle>
            <extension>2145</extension>
        </employee>
    </accounts_dept>
    <engineering_dept>
        <employee type='PT'>
            <name>Sue Jones</name>
            <extension>2375</extension>
        </employee>
    </engineering_dept>
</acme_org>
```

A *tag* is the text between the left and right angle brackets. <acme_org>, <employee> and <name> are examples of tags. There are starting tags, such as <name>, and ending tags, such as </name>. The above XML fragment describes the Acme organization with two employees, John Smith and Sue Jones.

An *element* is the starting tag, ending tag, and everything in between. In the sample above, the <employee> element contains three child elements: <name>, <cubicle>, and <extension>.

An *attribute* is a name-value pair inside the starting tag of an element. In this example, **type='PT'** is an attribute of the <employee> element;

In the above example, the tag *name* is an element of *employee*. Each element has a value. An example of a *name* element's value is the string "John Smith".

## Using XML Functions

The next sections provide examples of how to work with data in an XML tree. Certain functions allow you to retrieve information, and others let you write information to an XML tree. These examples use the following XML tree containing the names and extensions of several employees in the *Acme* organization.

```
<acme_org>
   <accounting_dept>
      <employee type='PT'>
         <name>John Smith</name>
         <extension>2145</extension>
      </employee>
   </accounting_dept>
   <engineering_dept>
      <employee type='PT'>
         <name>Sue Jones</name>
         <extension>2375</extension>
      </employee>
   </engineering_dept>
</acme_org>
```

### Reading Information from an XML Tree

The functions which read information from an XML tree are:

| | |
|---|---|
| **lr_xml_extract** | Extracts XML string fragments from an XML string. |
| **lr_xml_find** | Performs a query on an XML string. |
| **lr_xml_get_values** | Retrieves values of XML elements found by a query. |

To retrieve a specific value through a query, you specify the tags of the parent and child nodes in a path format.

For example, to retrieve an employee name in the Accounting department, use the following string:

```
lr_xml_get_values("XML={XML_Input_Param}",
"ValueParam=OutputParam",
"Query=/acme_org/accounting_dept/employee/name",
LAST);
```

The Execution log window (with Extended logging enabled) shows the output of this function:

```
Output:
Action.c(20): "lr_xml_get_values" was successful, 1 match processed
Action.c(25): Query result = John Smith
```

## Writing to an XML Structure

The functions which write values to an XML tree are:

| | |
|---|---|
| **lr_xml_delete** | Deletes fragments from an XML string. |
| **lr_xml_insert** | Inserts a new XML fragment into an XML string. |
| **lr_xml_replace** | Replaces fragments of an XML string. |
| **lr_xml_set_values** | Sets the values of XML elements found by a query. |
| **lr_xml_transform** | Applies Extensible Stylesheet Language (XSL) transformation to XML data. |

The most common *writing* function is **lr_xml_set_values** which sets the values of specified elements in an XML string. The following example uses **lr_xml_set_values** to change the phone extensions of two *employee* elements in an XML string.

First, we save the XML string to a parameter called *XML_Input_Param*. We want two values to be matched and substituted, so we prepare two new parameters, *ExtensionParam_1* and *ExtensionParam_2*, and set their values to two new phone extensions, 1111 and 2222.

**lr_xml_set_values** contains the argument "ValueName=ExtensionParam", which picks up the values of *ExtensionParam_1* and *ExtensionParam_2*. The current extensions of the two employees are substituted with the values of these parameters, 1111 and 2222. The value of *OutputParam* is then evaluated proving that the new phone extensions were in fact substituted.

```
Action() {

    int i, NumOfValues;
    char buf[64];

    lr_save_string(xml_input, "XML_Input_Param"); // Save input as
parameter
    lr_save_string("1111", "ExtensionParam_1");
    lr_save_string("2222", "ExtensionParam_2");

    lr_xml_set_values("XML={XML_Input_Param}",
        "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
        "SelectAll=yes", "Query=//extension", LAST);

    NumOfValues= lr_xml_get_values("XML={NewXmlParam}",
        "ValueParam=OutputParam", "Query=//extension",
        "SelectAll=yes", LAST);

    for (i = 0; i < NumOfValues; i++) {/* Print the multiple values of Multi-
Param */

        sprintf(buf, "Retrieved value %d : {OutputParam_%d}", i+1, i+1);
        lr_output_message(lr_eval_string(buf));
    }

    return 0;
}
Output:
Action.c(40): Retrieved value 1: 1111
Action.c(40): Retrieved value 2: 2222
```

# Specifying XML Function Parameters

Most XML API functions require that you specify the **XML element** and a **query**. You can also indicate if you want to retrieve all results or a single one.

### Defining the XML Element

For defining the XML element to query, you can specify a literal string of the XML element, or a parameter that contains the XML. The following example shows the XML input string defined as a literal string:

"XML=<employee>JohnSmith</employee>

Alternatively, the **XML** string can be a parameter containing the XML data. For example:

"XML={EmployeeNameParam}"

### Querying an XML Tree

Suppose you want to find a value within an XML tag, for example, an employee's extension. You formulate a query for the desired value. The query indicates the location of the element and which element you want to retrieve or set. The path that you specify limits the scope of the search to a specific tag. You can also search for all elements of a specific type under all nodes below the root.

For a specific path, use "Query=/*full_xml_path_name*/*element_name*"

For the same element name under all nodes, use "Query=//*element_name*"

In the VuGen implementation of XML functions, the scope of a query is the entire XML tree. The tree information is sent to the Vuser API functions as the value of the *xml* argument.

### Multiple Query Matching

When you perform a query on an XML element, by default VuGen returns only the first match. To retrieve multiple values from a query, you specify the "SelectAll=yes" attribute within your functions. VuGen adds a suffix of *_index* to indicate multiple parameters. For example, if you defined a parameter by the name *EmployeeName*, VuGen creates *EmployeeName_1*, *EmployeeName_2*, *EmployeeName_3*, and so on.

```
lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);
```

With functions that *write* to a parameter, the values written to the parameter can then be evaluated. For example, the following code retrieves and prints multiple matches of a query:

```
NumOfValues = lr_xml_get_values("Xml={XmlParam}", "Query=//name",
        "SelectAll=yes", "ValueParam=EmployeeName", LAST);
```

For functions that *read* from parameters, the values of the parameters must be pre-defined. The parameter must also use the convention *ParamName_IndexNumber*, for example *Param*_1, *Param_2*, *Param*_3, and so on. This collection of parameters is also known as a parameter set.

In the following example, **lr_xml_set_values** reads values from the parameter set and then uses those values in the XPath query. The parameter set that represents the employee extensions, is called *ExtensionParam*. It has two members: *ExtensionParam_1* and *ExtensionParam_2*. The **lr_xml_set_values** function queries the XML input string and sets the value of the first match to 1111 and the second match to 2222.

```
lr_save_string("1111", "ExtensionParam_1");
lr_save_string("2222", "ExtensionParam_2");

lr_xml_set_values("XML={XML_Input_Param}",
      "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
      "SelectAll=yes", "Query=//extension", LAST);
```

## Working with XML Attributes

VuGen contains support for attributes. You can use a simple expression to manipulate attributes of XML elements and nodes, just as you can manipulate the elements themselves. You can modify the desired attribute or only attributes with specific values.

In the following example, **lr_xml_delete** deletes the first *cubicle* element with the *name* attribute.

```
lr_xml_delete("Xml={ParamXml}",
                        "Query=//cubicle/@name",
                            "ResultParam=Result",
                        LAST
    );
```

In the next example, **lr_xml_delete** deletes the first cubicle element with a *name* attribute that is equal to *Paul*.

```
lr_xml_delete("Xml={ParamXml}",
            "Query=//cubicle/@name="Paul",
            "ResultParam=Result",
            LAST
    );
```

## Structuring an XML Script

Initially, you create a new script in your preferred protocol. You can record a session in that protocol, or you may program the entire script without recording. Structure the Actions section of the script as follows:

- XML input declaration
- The Actions section

The **XML input section** contains the XML tree that you want to use as an input variable. You define the XML tree as a **char** type variable. For example:

```
char *xml_input=
“<acme_org>”
      "<employee>"
           " <name>John Smith</name>"
           "<cubicle>227</cubicle>"
           "<extension>2145</extension>"
      “</employee>”
      "<employee>"
           "<name>Sue Jones</name>"
           "<cubicle>227</cubicle>"
           "<extension>2375</extension>"
      "</employee>"
"</acme_org>";
```

The **Action section** contains the evaluation of the variables and queries for the element values. In the following example, the xml input string is evaluated using **lr_save_string**. The input variable is queried for employee names and extensions.

```
Action() {

      /* Save the input as a parameter.*/
      lr_save_string(xml_input, "XML_Input_Param");

      /* Query 1 - Retrieve an employee name from the specified element.*/
      lr_xml_get_values("XML={XML_Input_Param}",
           "ValueParam=OutputParam",
           "Query=/acme_org/employee/name", LAST);

      /* Query 2 - Retrieve an extension under any path below the root.*/
      lr_xml_get_values("XML={XML_Input_Param}",
           "ValueParam=OutputParam",
           "Query=//extension", LAST);

      return 0;
}
```

## Enhancing a Recorded Session

You can prepare an XML script by recording a session and then manually adding the relevant XML and Vuser API functions.

The following example illustrates how a recorded session was enhanced with Vuser API functions. Note that the only function that was recorded was **web_submit_data**, which appears in bold.

The first section contains the XML input declaration of the variable SoapTemplate, for a SOAP message:

```
#include "as_web.h"

// SOAP message
const char*pSoapTemplate=
        "<soap:Envelope xmlns:soap=\"http://schemas.xml-
soap.org/soap/envelope/\">"
                "<soap:Body>"
                    "<SendMail xmlns=\"urn:EmailIPortTypeInft-IEmailSer-
vice\"/>"
                "</soap:Body>"
        "</soap:Envelope>";
```

The following section represents the actions of the user:

```
Action1()
{
      // get response body
      web_reg_save_param("ParamXml", "LB=", "RB=", "Search=body",
LAST);

      // fetch weather by HTTP GET
      web_submit_data("GetWeather",
                        "Action=http://glkev.net.innerhost.com/glkev_ws/
                              WeatherFetcher.asmx/GetWeather",
                           "Method=GET",
                           "EncType=",
                           "RecContentType=text/xml",
                           "Referer=http://glkev.net.innerhost.com
                          /glkev_ws/WeatherFetcher.asmx?op=GetWe
ather",
            "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}</Tempera-
ture>"
                                        "<Humidity>{ParamHumid}</Humid-
ity>"
                                        "<Conditions>{ParamCond}</Condi-
tions>"
                                    "</Weather>",
                                LAST
    );

    lr_output_message(lr_eval_string("***** Time = {ParamTime}, Tem-
perature =                                {ParamTemp}, "
                                    "Humidity = {ParamHumid}, Condi-
tions =                                {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. Fahr-
enheit\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</Fro-
mAddress>"
                                       "<ToAddress>support@merc-int.com</ToAd-
dress>"
                                       "<ASubject>Weather Forecast</ASubject>"
                                       "<MsgBody/>",
                              LAST
      );

//
//          "<soap:Envelope xmlns:soap=\"http://schemas.xml-
soap.org/soap/envelope/\">"
//                    "<soap:Body>"
//                        "<SendMail xmlns=\"urn:EmailIPortTypeInft-IEmailSer-
vice\"/>"
//                                       "<FromAddress>taurus@merc-int.com</Fro-
mAddress>"
//                                       "<ToAddress>support@merc-int.com</ToAd-
dress>"
//                                       "<ASubject>Weather Forecast</ASubject>"
//                                       "<MsgBody/>"
//                        "</SendMail>"
//                    "</soap:Body>"
//          "</soap:Envelope>";
//

      // Insert actual forecast text
      lr_xml_set_values("Xml={ParamRequest}",
                                 "ResultParam=ParamRequest",
                                 "Query=Body/SendMail/MsgBody",
                                 "ValueParam=ParamForecast",
                                 LAST);
```

```
	// Add header for SOAP
	web_add_header("SOAPAction", "urn:EmailIPortTypeInft-IEmailSer-
vice");

	// 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/IEmailser-
vice",
		"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;
}
```

# 78

# 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
- Adding Additional C Language Keywords
- Examining Replay Output
- Debugging Database Applications
- Working with Oracle Applications
- Solving Common Problems with Oracle 2-Tier Vusers
- Two-tier Database Scripting Tips
- Running PeopleSoft-Tuxedo Scripts

## General Debugging Tip

VuGen can be used as a regular text editor. You can open any text file in it and edit it. When an error message is displayed during replay in the output window below, you can double click on it and VuGen jumps the cursor to the line of the test that caused the problem. You can also place the cursor on the error code and press F1 to view the online help explanation for the error code.

## Using C Functions for Tracing

You can use the C interpreter trace option (in version 230 or higher) to debug your Vuser scripts. The ci_set_debug statement allows trace and debug to be turned on and off at specific points in the script.

**ci_set_debug**(ci_this_context, int debug, int trace);

For example, you could add the following statements to your script:

```
ci_set_debug(ci_this_context, 1, 1) /* turn ON trace & debug */
ci_set_debug(ci_this_context, 0, 0) /* turn OFF trace & debug */
```

## Adding Additional C Language Keywords

When you run a C script in VuGen, its parser uses the built-in C interpreter to parse the functions in the script. You can add keywords that are not part of the standard parser's library. By default, several common C++ keywords are added during installation, such as *size_t* and *DWORD*. You can edit the list and add additional keywords for your environment.

**To add additional keywords:**

1. Open the vugen_extra_keywords.ini file, located in your machine's <Windows> or <Windows>/System directory.
2. In the EXTRA_KEYWORDS_C section, add the desired keywords for the C interpreter.

The file has the following format:

```
[EXTRA_KEYWORDS_C]
FILE=
size_t=
WORD=
DWORD=
LPCSTR=
```

## Examining Replay Output

Look at the replay output (either from within VuGen, or the file **output.txt** representing the output of the VuGen driver). You may also change the run-time settings options in VuGen to select more extensive logging in order to obtain a more detailed log output of the replayed test.

## Debugging Database Applications

The following tips apply to database applications only (Oracle, ODBC, Ctlib):

- Generating Debugging Information
- Examining Compiler Information
- Code Generation Information
- Preprocessing and Compilation Information

### Generating Debugging Information

---

**Note:** You can now set options to view most of the information described in this section using VuGen's user interface.

---

VuGen contains an inspector "engine." You can force VuGen recorder to create "inspector" output by editing *\WINDOWS_DIR\vugen.ini* as follows:

```
[LogMode]
EnableAscii=ASCII_LOG_ON
```

When this option is enabled, VuGen creates a file, **vuser.asc** in the Data directory at the end of the recording. Note that this option should be used for debugging purposes only. This output file can become very large (several MB) and have serious effects on machine performance and disk space.

For cases like ODBC-based applications, it is possible to configure the ODBC Administrator (located in the Windows Control Panel) to provide a similar trace output. Open the ODBC options, and select 'Trace ODBC calls' to ON. Similarly the ODBC Developer Kit provides a Spy utility for call tracing.

To enable further debug information, add the following section to the \WINDOWS_DIR\vugen.ini file:

```
[INSPECTOR]
TRACE_LEVEL=3
TRACE_FILENAME=c:\tmp\sqltrace.txt
```

The file (sqltrace.txt) will include useful internal information regarding the hooking calls made during recording. The trace_level is between 1 and 3, with 3 representing the most detailed debug level. Note that in VuGen versions 5.02 and higher, you can set the trace level from the user interface.

### Examining Compiler Information

You can view information about each stage of code generation, preprocessing and compilation to determine the source of any errors.

### Code Generation Information

Look at the **vuser.log** file under the Data directory. This file, which contains a log of the code generation phase, is automatically created at the end of every lrd recording (i.e. all database protocols).

The following is an example of a log file:

```
lrd_init: OK
lrd_option: OK
lrd_option: OK
lrd_option: OK
Code generation successful
lrd_option: OK
lrd_end: OK
```

If any of the messages are not OK or successful, then a problem occurred during the code generation.

### Preprocessing and Compilation Information

During runtime, VuGen displays information about both the preprocessing and compilation processes.

## Working with Oracle Applications

Oracle Applications is a two-tier ("fat" client) packaged application, made up of 35 different modules (Oracle Human Resources, Oracle Financials, and so forth).

There are a number of issues that you should be aware of while recording and replaying Vusers for Oracle Applications:

- A typical script contains thousands of events, binds and assigns.
- A typical script has many db connections per user session.
- scripts almost always require correlated queries.
- Oracle Applications' clients are 16-bit only (developed with Oracle Developer 2000). This means that for debugging, if you don't have the Oracle 32bit client, you need to use VuGen's Force 16-bit options.

When a new window is created, the application retrieves an .xpf file from the file system for display. Currently, VuGen does not take this into consideration since it records at the client/server level. Therefore, there is a fairly significant inaccuracy in performance measurements since in most cases performance problems are related to the network bottleneck between clients and file server. We are currently thinking about this problem and how, if at all, to solve it.

## Solving Common Problems with Oracle 2-Tier Vusers

This section contains a list of common problems that you may encounter while working with Oracle Vusers, and suggested solutions.

### ORA-20001 and ORA-06512

Errors ORA-20001 and ORA-06512 appear during replay when the lrd_stmt contains the pl/sql block: fnd_signon.audit_responsibility(...)

This statement fails during replay because the sign-on number is unique for each new connection.

#### Solution

In order to solve this problem you need to use the new correlation tool for the sign-on number. This is second assigned value in the statement.

After you scan for possible values to correlate, highlight the value of the second lrd_assign_bind() for the failed statement. Note that the values in the "correlated query" window may not appear in the same order as the actual recorded statements.

The grid containing the substitution value should appear after the lrd_stmt which contains the pl/sql block: fnd_signon.audit_user(...).

---

**Note:** Since the sign-on number is unique for every connection, you need to use correlation for each new connection that you record.

---

### Example of Solution

The following statement failed in replay because the second value, "1498224" is the unique sign-on number for every new connection.

```
lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"
       "; end;", -1, 1, 1, 0);
   lrd_assign_bind(Csr6, "s", "D", &s_D216, 0, 0, 0);
   lrd_assign_bind(Csr6, "l", "1498224", &l_D217, 0, 0, 0);
   lrd_assign_bind(Csr6, "f", "1", &f_D218, 0, 0, 0);
   lrd_assign_bind(Csr6, "a", "810", &a_D219, 0, 0, 0);
   lrd_assign_bind(Csr6, "r", "20675", &r_D220, 0, 0, 0);
   lrd_assign_bind(Csr6, "t", "Windows PC", &t_D221, 0, 0, 0);
   lrd_assign_bind(Csr6, "p", "", &p_D222, 0, 0, 0);
   lrd_exec(Csr6, 1, 0, 0, 0, 0);
```

The sign-on number can be found in the lrd_stmt with "fnd_signon.audit_user". The value of the first placeholder "a" should be saved. The input of "a" is always "0" but the output is the requested value.

### Modified code:

```
 lrd_stmt(Csr4, "begin fnd_signon.audit_user(:a,:l,:u,:t,:n,:p,:s); end;", -1, 1, 1, 0);
   lrd_assign_bind(Csr4, "a", "0", &a_D46, 0, 0, 0);
   lrd_assign_bind(Csr4, "l", "D", &l_D47, 0, 0, 0);
   lrd_assign_bind(Csr4, "u", "1001", &u_D48, 0, 0, 0);
   lrd_assign_bind(Csr4, "t", "Windows PC", &t_D49, 0, 0, 0);
   lrd_assign_bind(Csr4, "n", "OraUser", &n_D50, 0, 0, 0);
   lrd_assign_bind(Csr4, "p", "", &p_D51, 0, 0, 0);
   lrd_assign_bind(Csr4, "s", "14157", &s_D52, 0, 0, 0);
   lrd_exec(Csr4, 1, 0, 0, 0, 0);

lrd_save_value(&a_D46, 0, 0, "saved_a_D46");
   Grid0(17);

lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"
       "; end;", -1, 1, 1, 0);
   lrd_assign_bind(Csr6, "s", "D", &s_D216, 0, 0, 0);
   lrd_assign_bind(Csr6, "l", "<saved_a_D46>", &l_D217, 0, 0, 0);
   lrd_assign_bind(Csr6, "f", "1", &f_D218, 0, 0, 0);
   lrd_assign_bind(Csr6, "a", "810", &a_D219, 0, 0, 0);
   lrd_assign_bind(Csr6, "r", "20675", &r_D220, 0, 0, 0);
```

```
lrd_assign_bind(Csr6, "t", "Windows PC", &t_D221, 0, 0, 0);
lrd_assign_bind(Csr6, "p", "", &p_D222, 0, 0, 0);
lrd_exec(Csr6, 1, 0, 0, 0, 0);
```

## Working with large numbers

Large numbers (NUMBER data type) sometimes appear in different format in the GRID and in the ASCII file. This difference makes it more difficult to identify numbers while searching for values to save for correlation.

For example, you could have a value appear as 1000003 in the grid, but as 1e+0006 in the Recording Log (ASCII file).

### Workaround

If you have an error during replay and the correlation tool cannot locate the value in previous results, look for this value in the other format in grid.

## ORA-00960

This error 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:

```
"lrdo.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960:
ambiguous column naming in select list".
```

### Workaround

Change the statement by adding an alias to at least one of the non-unique columns, thus mapping it to a new unique name. For example:

```
lrd_stmt(Csr9,"SELECT UOM_CODE,UOM_CODE second, DESCRIPTION
FROM"
      "MTL_UNITS_OF_MEASURE "
      "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
      "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

Alternate Workaround: remove ORDER BY from the lrd statement.

## ORA-2002

Error 2002 appears when you try to use an unopened cursor. It occurs when you replay a user more than one iteration and you recorded into more than one section of the script.

Specifically, if a cursor is opened in the vuser_init section and closed in the Actions section, then you will encounter this error on the second iteration if you try to use the cursor. This is because it was closed but not re-opened.

For example: You have *lrd_open_cursor* in the vuser_init section and *lrd_close_cursor* in the Actions section. If you replay this user more than one iteration, you are going to get an error in the second iteration because you try using an unopened cursor (it was closed in first iteration, but not re-opened in the second).

### Workaround

The easiest way to solve this is to move the **lrd_close_cursor** or/and **lrd_close_connection** of the problem cursor to the *vuser_end* section.

**Database Protocols (lrd)**

Replay of recorded asynchronous operations is not supported.

## Wrong Client Version

You may receive an error message when running the wrong Oracle client version:

"Error: lrdo_open_connection: "olog" LDA/CDA return-code_019: unable to allocate memory in the user side"

### Workaround

You need to modify the library information in he *lrd.ini* file, located in the your product's bin directory. This file contains the settings that indicate which version of Mercury's database support is loaded during recording or replay. The file contains a section for each type of host. For example, the following section of the *lrd.ini* file is for Oracle on HP/UX:

```
[ORACLE_HPUX]
;816=liblrdhpo816.sl
;81=liblrdhpo81.sl
;80=liblrdhpo80.sl
73=liblrdhpo73.sl
72=liblrdhpo72.sl
```

These settings indicate that Vusers should use the Mercury library liblrdhpo816.sl if the client uses Oracle 8.1.6, liblrdhpo81.sl for Oracle 8.1.5, and so on.

During replay on UNIX, the settings in the lrd *ini* file must indicate the correct version of the database to use. Suppose it is necessary to replay a Vuser for HP/UX using Oracle 8.1.5. In that case the previous lines for other versions of Oracle should be commented out with a ";" at the beginning of the line.

This section of the lrd.ini file will now look like:

```
[ORACLE_HPUX]
;816=liblrdhpo816.sl
81=liblrdhpo81.sl
;80=liblrdhpo80.sl
73=liblrdhpo73.sl
72=liblrdhpo72.sl
```

You also may need to make a change for Win32 if the application does not use the DLL mentioned in the lrd.ini file. For example, PowerBuilder 6.5 uses Oracle 8.0.5, but it uses the ora803.dll, not the ora805.dll. In that case, either comment out the 805 and 804 sections of the ORACLE_WINNT section, or change the 805 section from:

```
805=lrdo32.dll+ora805.dll
```

to

```
805=lrdo32.dll+ora803.dll
```

## Two-tier Database Scripting Tips

The following section offers solutions for two-tier database scripts. For Siebel specific solutions, see "Siebel-specific Scripting Tips" on page 1152.

**Question 1**: Why does the script fail when it is data driven, while the same values work with the application itself?

**Answer:** The failure may be a result of trailing spaces in your data values. Even though the data values that you type directly into the GUI are probably truncated, you should manually eliminate them from your data file. Tab-delimited files can hide trailing spaces and therefore obscure problems. In general, comma-delimited files are recommended. You can view the files in Excel to see if things are correct.

**Question 2**: Why does an SQL error of an invalid cursor state occur on the second iteration?

**Answer:** The **lrd_close_cursor** function may not have been generated or it may be in the *end* section instead of the *action* section. You will need to add a cursor close function or move it from the *end* section to make the script iterate successfully.

Opening a new cursor may be costly in terms of resources. Therefore, it is recommended that you only open a cursor once in the *actions* section during the first iteration. You can then add a new parameter that contains the iteration number as a string by using the Iteration Number type. Call this parameter *IterationNum*. Then, inside the *actions* section replace a call to open a new cursor like

```
lrd_open_cursor(&Csr1, Con1, 0);
```

with

```
if (!strcmp(lr_eval_string("<IterationNum>"), "1"))
   lrd_open_cursor(&Csr1, Con1, 0);
```

**Question 3**: How can I fix code produced by VuGen that will not compile because of data declarations in the *vdf.h* file?

**Answer:** The problem, most likely, is an SQL data type that is not supported by VuGen. For Microsoft SQL, you can often work around this issue by replacing the undefined error message in *vdf.h* with "DT_SZ" (null terminated string). Although this is not the actual datatype, VuGen can compile the script correctly. Please report the problem and send the original script to customer support.

**Question 4**: What is the meaning of LRD Error 2048?

**Answer:** VuGen is failing because it is trying to bind a variable with a longer length than what was allocated during recording. You can correct this by enlarging the variable definition in *vdf.h* to receive a longer string back from the database. Search this file for the unique numeric identifier. You will see its definition and length. The length is the third element in the structure. Increase this length as required and the script will replay successfully.

For example, in the following script, we have:

```
lrd_assign(&_2_D354, "<ROW_ID>", 0, 0, 0);
```

In *vdf.h*, we search for *_2_D354* and find

```
static LRD_VAR_DESC _2_D354 = {
        LRD_VAR_DESC_EYECAT, 1, 10, LRD_BYTYPE_ODBC,
        {0 ,0, 0}, DT_SZ, 0, 0, 15, 12};
```

We change it to:

```
static LRD_VAR_DESC _2_D354 = {
        LRD_VAR_DESC_EYECAT, 1, 12, LRD_BYTYPE_ODBC,
        {0,0, 0}, DT_SZ, 0, 0, 15, 12};
```

The complete definition of LRD_VAR_DESC appears in *lrd.h*. You can find it by searching for typedef struct LRD_VAR_DESC.

**Question 5**: How can I obtain the number of rows affected by an UPDATE, INSERT or DELETE when using ODBC and Oracle?

**Answer:** You can use **lrd** functions to obtain this information. For ODBC, use **lrd_row_count**. The syntax is:

```
int rowcount;
   .
   .
   .
lrd_row_count(Csr33, &rowcount, 0);
```

Note that **lrd_row_count** must immediately follow the pertinent statement execution.

For Oracle you can use the fourth argument of **lrd_exec**.

```
lrd_exec(Csr19, 1, 0, &rowcount, 0, 0);
```

If you are using Oracle's OCI 8, you can use the fifth argument of **lrd_ora8_exec**.

```
lrd_ora8_exec(OraSvc1, OraStm3, 1, 0, &uliRowsProcessed, 0, 0, 0, 0, 0);
```

**Question 6**: How can I avoid duplicate key violations?

**Answer:** Occasionally, you will see a duplicate key violation when performing an Insert. You should be able to find the primary key by comparing two recordings to determine the problem. Check whether this or earlier UPDATE or INSERT statement should use correlated queries. You can use the data dictionary in order to find the columns that are used in the violated unique constraint.

In Oracle you will see the following message when a unique constraint is violated:

```
ORA-00001: unique constraint (SCOTT.PK_EMP) violated
```

In this example SCOTT is the owner of the related unique index, and PK_EMP is the name of this index. Use SQL*Plus to query the data dictionary to find the columns. The pattern for this query is:

```
select column_name from all_ind_columns where index_name = '<IndexName>
and index_owner = '<IndexOwner>';

select column_name from all_ind_columns where index_name = 'PK_EMP' and
index_owner = 'SCOTT';
```

Since the values inserted into the database are new, they might not appear in earlier queries, but they could be related to the results of earlier queries, such as one more than the value returned in an earlier query.

For Microsoft SQL Server you will see one of these messages:

```
Cannot insert duplicate key row in object 'newtab' with unique index 'IX_newtab'.

Violation of UNIQUE KEY constraint 'IX_Mark_Table'. Cannot insert duplicate
key in object 'Mark_Table'.

Violation of PRIMARY KEY constraint 'PK_NewTab'. Cannot insert duplicate key
in object 'NewTab'.
```

You can use the Query Analyzer to find out which columns used by the key or index. The pattern for this query is:

```
select C.name
    from sysindexes A, sysindexkeys B, syscolumns C
    where C.colid = B.colid and C.id = B.id and
    A.id = B.id and A.indid = B.indid
    and A.name = '<IndexName>' and A.id = object_id('<TableName>')

select C.name
    from sysindexes A, sysindexkeys B, syscolumns C
    where C.colid = B.colid and C.id = B.id and
    A.id = B.id and A.indid = B.indid
    and A.name = 'IX_newtab' and A.id = object_id('newtab')
```

For DB2 you might see the following message:

```
SQL0803N One or more values in the INSERT statement, UPDATE statement,
or foreign key update caused by a DELETE statement are not valid because
they would produce duplicate rows for a table with a primary key, unique
constraint, or unique index. SQLSTATE=23505
```

If you still encounter problems, be sure to check the number of rows changed for Updates and Inserts for both recording and replay. Very often, an UPDATE fails to change any rows during replay, because the WHERE clause was not satisfied. This does not directly result in an error, but it causes a table not to be properly updated, and can cause a later SELECT to choose the wrong value when correlating the query.

Also verify that there are no problems during multi-user replay. In certain instances, only one user will successfully perform an UPDATE. This occurs with Siebel, where it is necessary to manually write a loop to overcome the problem.

**Question 7**: The database does not appear to be modified after replaying a script which should have modified the database.

**Answer:** Through the user application's UI, check if the updated values appear when trying to see the current data accessible to the application. If the values have not been updated, you need to determine they were not changed. Possibly, an UPDATE statement changed one or more rows when the application was recorded, and did not change any during replay.

Check these items:

- **Verify statement:** If there is a WHERE clause in the UPDATE statement, verify that it is correct.
- **Check for correlations:** Record the application twice and compare the UPDATE statements from each of the recordings to make sure that the necessary correlations were performed.
- **Check the total number of rows:** Check the number of rows that were changed after the UPDATE. For Oracle, this information is stored in the fourth parameter of **lrd_exec**. For ODBC, use **lrd_row_count** to determine the number of rows updated. You can also add code to your script that prints the number of rows that were updated. If this value is 0, the UPDATE failed to modify the database.
- **Check the SET clause:** Check the SET clause of the UPDATE statement. Make sure that you correlated any necessary values here instead of hard-coding them. You can see this by comparing two recordings of the UPDATE.

It certain cases, the UPDATE works when replaying one Vuser, but not for multiple Vusers. The UPDATE of one Vuser might interfere with that of another. Parameterize each Vuser so that each one uses different values during the UPDATE, unless you want each vuser to update with the same values. In this case try adding retry logic to perform the UPDATE a second time.

**Question 8:** How do I avoid the unique column name error when replaying a statement recorded with an Oracle Application. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "
        "MTL_UNITS_OF_MEASURE "
        "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
        "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

The following error message was issued:

```
"lrdo.c/fjParse:  "oparse" ERROR return-code=960, oerhms=ORA-00960:
ambiguous column naming in select list".
```

**Answer:** Change the statement by adding an alias to at least one of the non-unique columns, thereby mapping it to a new unique name. For example:

```
lrd_stmt(Csr9,"SELECT UOM_CODE,UOM_CODE second, DESCRIPTION
FROM"
        "MTL_UNITS_OF_MEASURE "
        "WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "
        "SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

## Siebel-specific Scripting Tips

This section offers solutions for Siebel database users. You should also see the previous section which discusses some general database scripting tips.

**Question 9**: Virtual users run fine in VuGen but fail in the Controller or 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.

VuGen automatically correlates the NEXT_SUFFIX and MODIFICATION_NUM fields of the S_SSA_ID table. During an UPDATE the MODIFICATION_NUM field is incremented by 1 and the NEXT_SUFFIX field is increased by 100 in base 36. However, VuGen does not add code in instances where a client could not obtain a new block of suffix values. As a result, the replay fails with a unique constraint error, when you attempt to insert new values into the database.

You must manually add code to each location in the script where a block of suffixes is obtained, in order to perform a retry if the first attempt fails. You can locate these places by searching for SiebelPreSave in the script. You must also add a *while* loop with code similar to the example below. This example only works for Oracle. For ODBC use **lrd_row_count** instead of using the fourth argument of **lrd_exec**.

```
unsigned long lRowUpdated;
int nAttempt;

…

// This loops until we successfully obtain a “next_suffix”
lRowUpdated = 0;
nAttempt=0;

while (lRowUpdated != 1) {

   nAttempt++;
   if (nAttempt > 1)
      lr_output_message (".....Next suffix retry %d", nAttempt);
   else
   {
      lrd_open_cursor(&Csr13, Con1, 0);
      lrd_stmt(Csr13, "SELECT\n T1.LAST_UPD,\n T1.CREATED_BY,\n "
         "T1.CONFLICT_ID,\n T1.CREATED,\n T1.NEXT_SUFFIX,\n "
         "T1.ROW_ID,\n T1.NEXT_PREFIX,\n T1.CORPORATE_PREFIX,\n "
         "T1.MODIFICATION_NUM,\n T1.NEXT_FILE_SUFFIX,\n      "
         "T1.LAST_UPD_BY\n   FROM \n SIEBEL.S_SSA_ID T1", -1, 1, 1, 0);
   }
   lrd_bind_cols(Csr13, BCInfo_D375, 0);
   lrd_exec(Csr13, 0, 0, 0, 0, 0);
```

```
   SiebelPreSave_1();
   lrd_fetch(Csr13, -1, 4, 0, PrintRow26, 0);
   GRID(26);
   SiebelPostSave_1();

   if (nAttempt > 1)
   {
    lrd_open_cursor(&Csr14, Con1, 0);
    lrd_stmt(Csr14,"\nUPDATE SIEBEL.S_SSA_ID SET\n LAST_UPD_BY=:1,\n "
    "NEXT_SUFFIX = :2,\n      MODIFICATION_NUM = :3,\n      LAST_UPD = "
    ":4\n WHERE\n ROW_ID = :5 AND MODIFICATION_NUM = :6\n", -1, 1,
    1, 0);
   }
   lrd_assign_bind(Csr14, "6", "<modification_num>", &_6_D376, 0,
                LRD_BIND_BY_NUMBER, 0);
   lrd_assign_bind(Csr14,"5", "0-11",&_5_D377,0,LRD_BIND_BY_NUMBER, 0);
   strcpy (szTimeAtNewButton, lr_eval_string("<Now>"));
   sprintf (szTimeStamp, "%s %s", lr_eval_string("<Today>"),
            szTimeAtNewButton);
   lr_save_string (szTimeStamp, "DateTimeStamp");
lrd_assign_bind(Csr14, "4", "<DateTimeStamp>", &_4_D378, 0,
      LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "3", "<next_modnum>", &_3_D379, 0,
      LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "2", "<next_suffix_x100>", &_2_D380, 0,
      LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "1", "1-1E1",&_1_D381,0,LRD_BIND_BY_NUMBER, 0);

// this update won't update any rows unless we successfully got our suffix
lrd_exec(Csr14, 1, 0, &lRowUpdated, 0, 0);
lrd_commit(0, Con1, 0);

}//while
      lr_output_message ("…Rows updated %ld", lRowUpdated);
```

**Question 10**: How can I find the correct value to correlate for a primary key?

**Answer:** Siebel tends to generate key values based on base 36 mathematical manipulations of *<next_suffix>*. Try comparing several recordings and try to determine the relationships. You can ignore date fields when correlating Siebel, since they do not seem to effect script replay.

**Question 11**: How can I solve an INSERT into S_SRV_REQ failure with a duplicate key violation?

**Answer:** The primary key is SR_NUM. Newer versions of VuGen automatically correlate insertions into this table, by using the function lrd_siebel_str2num, which converts the NEXT_SUFFIX value of the S_SSA_ID table from base 36 to the base 10 equivalent. Older versions of VuGen might not handle this correlation correctly.

**Question 12**: VuGen does not automatically perform all the correlations I need in order to replay my script correctly. How can I add the missing correlations?

**Answer:** Currently VuGen only saves the values of the NEXT_SUFFIX and MODIFICATION_NUM columns from the S_SSA_ID table and replaces them with parameters when they are used later in the script. You may need to add some additional correlations manually. The correlation code in the **SiebelPreSave** and **SiebelPostSave** functions in the *print.inl* file can serve as an example of how to correlate specific values once you determine what needs to be correlated.

- Sometimes the NEXT_FILE_SUFFIX and MODIFICATION_NUM columns are chosen from the S_SSA_ID table. In this case, an UPDATE statement updates the NEXT_FILE_SUFFIX by adding one to this string in base 36, and one to the MODIFICATION_NUM. The value of the NEXT_FILE_SUFFIX will often be inserted in the FILE_REV_NUM field of a table. Often the name of this table ends with the *_ATT* suffix, to indicate that it is an attachment.
- Whenever Siebel performs an UPDATE statement, there is a MODIFICATION_NUM column that is incremented by one. VuGen only generates this correlation automatically for the S_SSA_ID table. You have to do it manually for other cases.

- Siebel refers to records according to their ID number. Siebel usually finds all records of a particular type (such as an agreement), and then later uses the ID number for a record when trying to update or delete an existing record of this type. You need to replace the ID number by a parameter during replay in order to generate a meaningful load test. The ID number has the form of one or more digits, a hyphen, followed by one or more alphanumeric characters, such as 1-QPF9. VuGen does not do this parameterization automatically, so you have to do it manually.
- If you find any other missing correlations or parameterizations, please notify customer support in order that Mercury Interactive can improve VuGen's support for Siebel.

## Running PeopleSoft-Tuxedo Scripts

To run PeopleSoft-Tuxedo Vusers with Tuxedo 7.x, you must change the library extension in the *mdrv.dat* file:

```
[PeopleSoft-Tuxedo]
WINNT_EXT_LIBS=lrt7.dll
```

# 79

# Advanced Topics

This chapter contains additional information for advanced users of VuGen.

- Files Generated During Recording
- Files Generated During Replay
- Running a Vuser from the Unix Command Line
- Specifying the Vuser Behavior
- Command Line Parameters
- Recording OLE Servers
- 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, and so on, from the development files.

   The following command line is used:

   ```
   cpp  -foptions.txt
   ```

4. The file **pre_cci.c** is created which is also a C file (**pre_cci.c** is defined in the **options.txt** file). The file **logfile.log** (also defined in **options.txt**) is created containing any output of this process. This file should be empty if there are no problems with the preprocessing stage. If the file is not empty then its almost certain that the next stage of compilation will fail due to a fatal error.

5. The **cci.exe** C compiler is now invoked to create a platform-dependent pseudo-binary file (.ci) to be used by the virtual user driver program that will interpret it at run-time. The cci takes the **pre_cci.c** file as input.

6. The file **pre_cci.ci** is created as follows:

   ```
   cci  -errout  c:\tmp\Vuser\logfile.log  -c  pre_cci.c
   ```

7. The file **logfile.log** is the log file containing output of the compilation.

8. The file **pre_cci.ci** is now renamed to **Vuser.ci**.

   Since the compilation can contain both warnings and errors, and since the driver does not know the results of this process, the driver first checks if there are entries in the **logfile.log** file. If there are, it then checks if the file **Vuser.ci** has been built. If the file size is not zero, it means that the cci has succeeded to compile - if not then compilation has failed and an error message will be given.

9. The relevant driver is now run taking both the **.usr** file and the **Vuser.ci** file as input. For example:

   ```
   mdrv.exe  -usr c:\tmp\Vuser\Vuser.usr  -out c:\tmp\Vuser  -file
   c:\tmp\Vuser\Vuser.ci
   ```

   The **.usr** file is needed since it tells the driver program which database is being used. From here it can then know which libraries need to be loaded for the run.

10. The **output.txt** file is created (in the path defined by the 'out' variable) containing all the output messages of the run. This is the same output as seen in both the VuGen runtime output window and the VuGen main lower window.

### Example of options.txt file

```
-DCCI
-D_IDA_XL
-DWINNT
-Ic:\tmp\Vuser   (name and location of Vuser include files)
-IE:\LRUN45B2\include   (name and location of include files)
-ec:\tmp\Vuser\logfile.log (name and location of output logfile)
 c:\tmp\Vuser\VUSER.c   (name and location of file to be processed)
```

### Example of Vuser.c file

```
#include "E:\LRUN45B2\include\lrun.h"
#include "c:\tmp\web\init.c"
#include "c:\tmp\web\run.c"
#include "c:\tmp\web\end.c"
```

## Running a Vuser from the Unix Command Line

VuGen includes a Unix shell script utility, *run_db_Vuser.sh*, that automatically performs the same operations as the virtual user but from the command line. It can perform each of the replay steps optionally and independently. This is a useful tool for debugging tests to be replayed on Unix.

Place the file *run_db_Vuser.sh* in the $M_LROOT/bin directory. To replay a Vuser type:

```
run_db_Vuser.sh   Vuser.usr
```

You can also use the following command line options:

| | |
|---|---|
| *-cpp_only* | This option will start the prepocessing phase. The output of this process is the file *'Vuser.c'*. |
| *-cci_only* | This option runs the compilation phase. The *'Vuser.c'* file is used as input, and the output produced is the *'Vuser.ci'* file. |

| | |
|---|---|
| *-exec_only* | This option runs the Vuser, by taking as input the *'Vuser.ci'* file and running it via the replay driver. |
| *-ci ci_file* | This option allows you to specify the name and location of a .ci file to be run. The second parameter contains the location of the .ci file. |
| *-out output_directory* | This option allows you to determine the location of any output files created throughout the various processes. The second parameter is the directory name and location. |
| *-driver driver_path* | This option allows you to specify the actual driver executable to be used for running the Vuser. By default the driver executable is taken from the settings in the VuGen.dat file. |

Note that only one of the first three options can be used at a time for running the run_db_vuser.

## Specifying the Vuser Behavior

Since VuGen creates the Vuser script and the Vuser behavior as two independent sources, you can configure user behavior without directly referencing the Vuser script, for example, wait times, pacing times, looping iterations, logging, and so forth. This feature lets you make configuration changes to a Vuser, as well as store several 'profiles' for the same Vuser script.

The *'Vuser.cfg'* file, by default, is responsible for defining this behavior - as specified in VuGen's Runtime settings dialog box. You can save several versions of this file for different user behavior and then run the Vuser script referencing the relevant *.cfg* file.

You can run the Vuser script with the relevant configuration file from a server machine. To do this, add the following to the Vuser command line:

```
-cfg c:\tmp\profile2.cfg
```

For information on command line parameters, see "Command Line Parameters" on page 1164.

Note that you cannot control the behavior file from VuGen. VuGen automatically uses the *.cfg* file with the same name as the Vuser. (You can, of course, rename the file to be *'Vuser.cfg'*). However, you can do this manually from the command line by adding the -cfg parameter mentioned above to the end of the driver command line.

---

**Note:** The Unix utility, *run_db_vuser*, does not yet support this option.

---

## Command Line Parameters

The Vusers can accept command line parameters when invoked. There are several Vuser API functions available to reference them (**lr_get_attrib_double**, and so on). In your environment, you can send command line parameters to the Vuser by adding them to the command line entry of the script window.

When running the Vuser from VuGen, you cannot control the command line parameters. You can do this manually, however, from the Windows command line by adding the parameters at the end of the line, after all the other driver parameters, for example:

```
mdrv.exe -usr c:\tmp\Vuser\Vuser.usr    -out c:\tmp\vuser
vuser_command_line_params
```

---

**Note:** The Unix utility, *run_db_vuser*, does not yet support this option.

---

# Recording OLE Servers

VuGen currently does not support recording for OLE applications. These are applications where the actual process is not launched by the standard process creation routines, but by the OLE Automation system. However, you can create a Vuser script for OLE applications based on the following guidelines.

There are two types of OLE servers: executables, and DLLs.

### DLL Servers

If the server is the DLL, it will eventually be loaded into the application process space, and VuGen will record the call to LoadLibrary. In this case, you may not even realize that it was an OLE application.

### Executable Servers

If the server is the executable, you must invoke the executable in the VuGen in a special way:

- First, determine which process actually needs to be recorded. In most cases, the customer knows the name of the application's executable. If the customer doesn't know the name of the application, invoke it and determine its name from the NT Task Manager.
- After you identify the required process, click **Start Recording** in VuGen. When prompted for the Application name, enter the OLE application followed by the flag "/Automation". Next, launch the user process in the usual way (not via VuGen). VuGen records the running OLE server and does not invoke another copy of it. In most cases, these steps are sufficient to enable VuGen to record the actions of an OLE server.
- If you still are experiencing difficulties with recording, you can use the *CmdLine* program to determine the full command line of a process which is not directly launched. (The program is available Patches section of the CSO web site, http://support.mercuryinteractive.com)

### Using CmdLine

In the following example, *CmdLine.exe* is used to determine the full command line for the process MyOleSrv.exe, which is launched by some other process.

**To determine its full command line:**

1. Rename *MyOleSrv.exe* to *MyOleSrv.orig.exe*.
2. Place *CmdLine.exe* in the same directory as the application, and rename it to *MyOleSrv.exe*.
3. Launch *MyOleSrv.exe*. It issues a popup with a message containing the complete command line of the original application, (including additional information), and writes the information into *c:\temp\CmdLine.txt*.
4. Restore the old names, and launch the OLE server, *MyOleSrv.exe*, from VuGen with the correct command line parameters. Launch the user application in a regular way - not through VuGen. In most cases, VuGen will record properly.

**If you still are experiencing difficulties with recording, proceed with the following steps:**

1. Rename the OLE server to MyOleSrv.1.exe, and CmdLine to MyOleSrv.exe.
2. Set the environment variables "CmdStartNotepad" and "CmdNoPopup" to 1. See “CmdLine Environment Variables” on page 1167 for a list of the CmdLine environment variables.
3. Start the application (not from VuGen). Notepad opens with the full command line. Check the command line arguments. Start the application several times and compare the command line arguments. If the arguments are the same each time you invoke the application, then you can reset the CmdStartNotepad environment variable. Otherwise, leave it set to "1".
4. In VuGen, invoke the program, MyOleSrv.1.exe with the command line parameters (use Copy/Paste from the Notepad window).
5. Start the application (not from within VuGen).

### CmdLine Environment Variables

You can control the execution of CmdLine through the following environment variables:

| | |
|---|---|
| **CmdNoPopup** | If set, the popup window will not appear. |
| **CmdOutFileName** | If set, and non-empty, CmdLine will attempt to create this file instead of c:\temp\CmdLine.txt. |
| **CmdStartNotepad** | If set, the output file will be displayed in the notepad (Best used with CmdNoPopup). |

## Examining the .dat Files

There are two .dat files used by VuGen: vugen.dat and mdrv.dat.

### vugen.dat

This vugen.dat file resides in the M_LROOT\dat directory and contains general information about VuGen, to be used by both the VuGen and the Controller or 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 seperated by a
comma>
ExtCmdLineConc=<extra command line (if the attr exists concatenate
value)>
ExtCmdLineOverwrite=<extra command line (if the attr exists overwrite
value)>
CallActionByNameFunc=<interpreter exec_action function>
GetFuncAddress=<interpreter get_location function>
RunLogicInitFunc=<action_logic init function>
RunLogicRunFunc=<action_logic run function>
RunLogicEndFunc=<action_logic end function>
```

For example, an Oracle NCA Vuser type is represented by:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp11i.dll
WIN95_EXT_LIBS=ncarp11i.dll
LINUX_EXT_LIBS=liboranca11i.so
SOLARIS_EXT_LIBS=liboranca11i.so
HPUX_EXT_LIBS=liboranca11i.sl
AIX_EXT_LIBS=liboranca11i.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api,HttpEngine
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
SecurityRequirementsFiles=oracle_nca.asl
SecurityMode=On
```

VuGen was designed to be able to handle a new Vuser type with no code modifications. You may, however, need to add a special View.

There is no generic driver supplied with VuGen, but you can customize one of the existing drivers. To use a customized driver, modify *mdrv.dat*. Add a line with the platform and existing driver, then add a new line with your customized driver name, in the format <*platform*>*_DLLS=*<*my_replay.dll name*>. For example, if your SAP replay dll is called SAPPLAY32.DLL, add the following two lines to the [sap] section of *mdrv.dat*:

```
WINNT=sapdrv32.exe
WINNT_DLLS=sapplay32.dll
```

## Adding a CFG file

You can optionally specify a configuration file to set the default Run-Time Settings for your protocol. You define it in the **LibCfgFunc** variable in the *mdrv.dat* file, or place one called *default.cfg* in the new protocols subdirectory under templates. A sample default.cfg follows.

```
[ThinkTime]
Options=NOTHINK
Factor=1
LimitFlag=0
Limit=1

[Iterations]
NumOfIterations=1
IterationPace=IterationASAP
StartEvery=60
RandomMin=60
RandomMax=90

[Log]
LogOptions=LogExtended
MsgClassData=0
MsgClassParameters=0
MsgClassFull=1
```

### Inserting an LRP file

In the dat/protocols directory, insert an *lrp* file which defines the protocol. This file contains the configuration information for the protocol in the Protocol, Template, VuGen, and API sections. Certain protocols may have additional sections, corresponding to the additional run-time setting options.

The **Protocol** section contains the name, category, description, and bitmap location for the protocol.

```
[Protocol]
Name=WAP
CommonName=WAP
Category=Wireless
Description=Wireless Application Protocol - used for Web-based, wireless
communication between mobile devices and content providers.
Icon=bitmaps\wap.bmp
Hidden=0
Single=1
Multi=0
```

The **Template** section indicates the name of the various sections of the script and the default test name.

```
[Template]
vuser_init.c=init.c
vuser_end.c=end.c
Action1.c=action.c
Default.usp=test.usp
@@TestName@@.usr=wap.usr
default.cfg=default.cfg
```

The **VuGen** section has information about the record and replay engines, along with the necessary DLLs and run-time files.

The **API** section contains information about the protocol's script API functions.

You can use one of the existing lrp files in the protocols directory as a base for your new protocol.

## Specifying a Template

After adding an *lrp* file, insert a subdirectory to *M_LROOT/*template with a name corresponding to the protocol name defined in the *lrp* file. In this subdirectory, insert a *default.cfg* file which defines the default settings for the general and run-time settings.

If you want to use a global header file for all of your protocol's scripts, add a file named *globals.h*. This file should contain an include statement which points to a header file for the new protocol. For example, the template/http subdirectory contains a file called *globals.h* which directs VuGen to the *as_web.h* file in the include directory:

```
#include #as_web.h"
```

# Part XVII

## Appendixes

# A

# Calling External Functions

When working with VuGen, you can call functions that are defined in external DLLs. By calling external functions from your script, you can reduce the memory footprint of your script and the overall run-time.

To call the external function, you load the DLL in which the function is defined.

You can load a DLL:

- locally—for one script, using the **lr_load_dll** function
- globally—for all scripts, by adding statements to the vugen.dat file

## Loading a DLL—Locally

You use the **lr_load_dll** function to load the DLL in your Vuser script. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

**To call a function defined in a DLL:**

1 Use the **lr_load_dll** function to load the DLL at the beginning of your script. Place the statement at the beginning of the *vuser_init* section. **lr_load_dll** replaces the **ci_load_dll** function.

Use the following syntax:

lr_load_dll(*library_name*);

Note that for UNIX platforms, DLLs are known as shared libraries. The extension of the libraries is platform dependent.

**2** Call the function defined in the DLL in the appropriate place within your script.

In the following example, the *insert_vals* function, defined in *orac1.dll*, is called, after the creation of the Test_1 table.

```
int LR_FUNC Actions(LR_PARAM p)
{
lr_load_dll("orac1.dll");

lrd_stmt(Csr1, "create table Test_1 (name char(15), id integer)\n", -1,
        1 /*Deferred*/, 1 /*Dflt Ora Ver*/, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);

/* Call the insert_vals function to insert values into the table. */
insert_vals();

lrd_stmt(Csr1, "select * from Test_1\n", -1, 1 /*Deferred*/, 1 /*Dflt Ora Ver*/,
0);
lrd_bind_col(Csr1, 1, &NAME_D11, 0, 0);
lrd_bind_col(Csr1, 2, &ID_D12, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_fetch(Csr1, -4, 15, 0, PrintRow14, 0);
…
```

---

**Note:** You can specify a full path for the DLL. If you do not specify a path, lr_load_library searches for the DLL using the standard sequence used by the C++ function, LoadLibrary on Windows platforms. On UNIX platforms you can set the LD_LIBRARY_PATH environment variable (or the platform equivalent). The **lr_load_dll** function uses the same search rules as *dlopen*. For more information, see the main pages for dlopen or its equivalent.

---

## Loading a DLL—Globally

You can load a DLL globally, to make its functions available to all your Vuser scripts. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

**To call a function defined in a DLL:**

1 Add a list of the DLLs you want to load to the appropriate section of the *mdrv.dat* file, located in your application's *dat* directory.

Use the following syntax,

*PLATFORM*_DLLS=*my_dll1*.dll, *my_dll2*.dll, …

replacing the word *PLATFORM* with your specific platform. For a list of platforms, see the beginning section of the *mdrv.dat* file.

For example, to load DLLs for Winsocket Vusers on an NT platform, add the following statement to the *mdrv.dat* file:

```
[WinSock]
ExtPriorityType=protocol
WINNT_EXT_LIBS=wsrun32.dll
WIN95_EXT_LIBS=wsrun32.dll
LINUX_EXT_LIBS=liblrs.so
SOLARIS_EXT_LIBS=liblrs.so
HPUX_EXT_LIBS=liblrs.sl
AIX_EXT_LIBS=liblrs.so
LibCfgFunc=winsock_exten_conf
UtilityExt=lrun_api
ExtMessageQueue=0
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
WINNT_DLLS=user_dll1.dll, user_dll2.dll, …
```

2 Call the function defined in the DLL in the appropriate place within your script.

# B

# Working with Foreign Languages

VuGen supports multilingual environments, allowing you to use languages other than English on native language machines when creating and running scripts.

This appendix describes:

- About Working with Foreign Languages
- Manually Converting String Encoding
- Converting String Encoding In Parameter Files
- Setting the String Encoding for Web Record and Replay
- Specifying a Language for the Accept-Language Header
- Protocol Limitations
- Quality Center Integration

## About Working with Foreign Languages

When working with languages other than English, the primary issue is ensuring that VuGen recognizes the encoding of the text during record and replay. The encoding applies to all texts used by the script. This includes texts in HTTP headers and HTML pages for Web Vusers, data in parameter files, and others.

Windows 2000 and higher lets you save text files with a specific encoding directly from Notepad: ANSI, Unicode, Unicode big endian, or UTF-8.

By default, VuGen works with the local machine encoding (ANSI). Some servers working with foreign languages, require you to work with UTF-8 encoding. To work against this server, you must indicate in the Advanced recording options, that your script requires UTF-8 encoding.

## Manually Converting String Encoding

You can manually convert a string from one encoding to another (UTF-8, Unicode, or locale machine encoding) using the **lr_convert_string_encoding** function. The syntax of the function is:

```
lr_convert_string_encoding(char * sourceString, char * fromEncoding, char * toEncoding, char * paramName)
```

The function saves the result string (including its terminating NULL) in the third argument, *paramName*. It returns a 0 on success and -1 on failure.

The format for the **fromEncoding** and **toEncoding** arguments are:

| | |
|---|---|
| LR_ENC_SYSTEM_LOCALE | NULL |
| LR_ENC_UTF8 | "utf-8" |
| LR_ENC_UNICODE | "ucs-2" |

In the following example, **lr_convert_string_encoding** converts “Hello world" from the system locale to Unicode.

```
Action()
{
   int rc = 0;
   unsigned long converted_buffer_size_unicode = 0;
   char              *converted_buffer_unicode = NULL;

   rc = lr_convert_string_encoding("Hello world", NULL,
LR_ENC_UNICODE, "stringInUnicode");
   if(rc < 0)
   {
      // error
   }
return 0;
}
```

In the Execution log, the output window shows the following information:

```
Output:
Starting action Action.
Action.c(7): Notify: Saving Parameter "stringInUnicode =
H\x00e\x00l\x00l\x00o\x00 \x00w\x00o\x00r\x00l\x00d\x00\x00\x00"
Ending action Action.
```

The result of the conversion is saved to the *paramName* argument.

## Converting String Encoding In Parameter Files

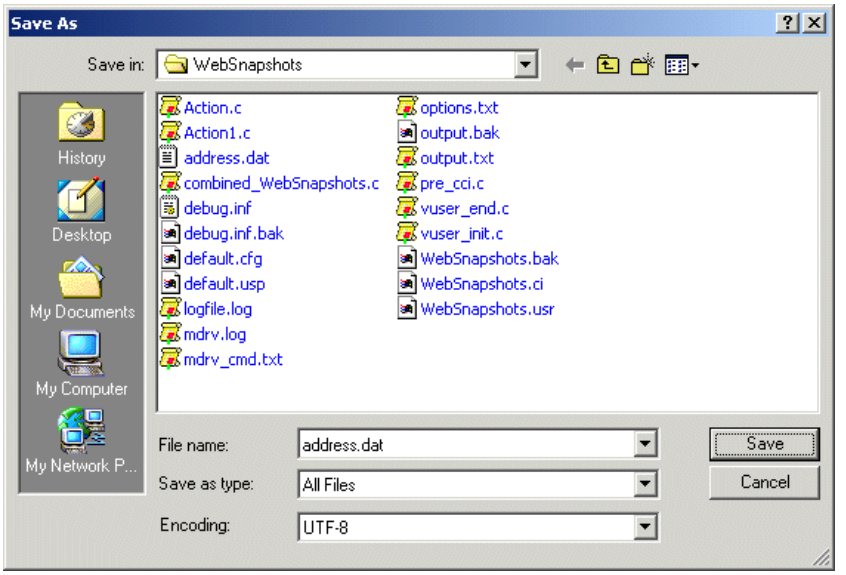
The parameter file contains the data for parameters that were defined in the script. This file, stored in the script’s directory, has a **.dat* extension. When running a script, Vusers use the data to execute actions with varying values.

By default, VuGen saves the parameter file with your machine’s encoding. When working with languages other than English, however, in cases where the server expects to receive the string in UTF-8, you may need to convert the parameter file to UTF-8. You can do this directly from Notepad, provided that you are working with Windows 2000 or higher.

**To apply UTF-8 encoding to a parameter file:**

1. Choose **Vuser** > **Parameter List** and view the parameter properties.
2. In the right pane, locate the parameter file in the **File path** box.
3. With the parameter table in view, click **Edit in Notepad**. Notepad opens with the parameter file in csv format.
4. In the **Save as type** box, select *All Files*.

   In the **Encoding** box, select *UTF-8* type encoding.




5. Click **Save**. Notepad asks you to confirm the overwriting of the existing parameter file. Click **Yes**.

   VuGen now recognizes the parameter file as UTF-8 text, although it still displays it in regular characters.

# Setting the String Encoding for Web Record and Replay

When working with Web or other Internet protocols, you can indicate the encoding of the Web page text for recording. The recorded site's language must match the operating system language. You cannot mix encodings in a single recording—for example, UTF-8 together with ISO-8859-1 or shift_jis.

This section discusses:

- Encoding Recording Option
- Manually Enabling Encoding
- Browser Configuration

### Encoding Recording Option

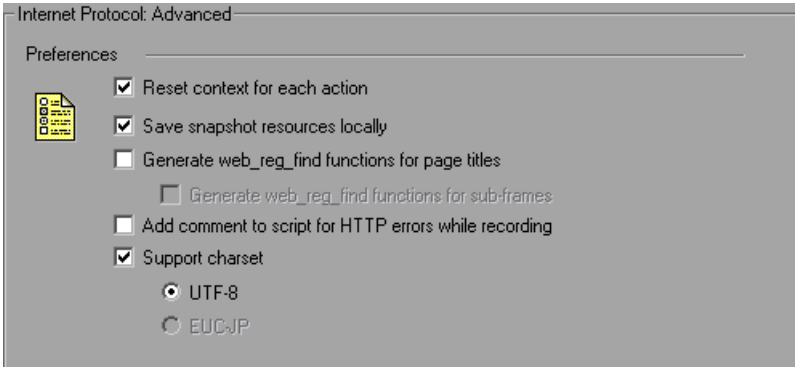
In order to be recognized as a non-English Web page, the page must indicate the charset in the HTTP header or in the HTML meta tag. Otherwise, VuGen will not detect the EUC-JP encoding and the Web site will not be recorded properly. To instruct VuGen to record non-English requests as **EUC-JP** or **UT-8**, select the appropriate option in the Recording Options dialog box, **Internet Protocol: Advanced** node.

- **UTF-8:** This option enables support for UTF-8 encoding. This instructs VuGen to convert non-ASCII UTF-8 characters to the encoding of your locale's machine in order to display them properly in VuGen.
- **EUC-JP:** For users of Japanese Windows, select this option to enable support for Web sites that use EUC-JP character encoding. This instructs VuGen to convert EUC-JP strings to the encoding of your locale's machine in order to display them properly in VuGen. VuGen converts all EUC-JP (Japanese UNIX) strings to the SJIS (Japanese Windows) encoding of your locale machine, and adds a **web_sjis_to_euc_param** function to the script (Kanji only).

Note that by selecting the **EUC-JP** or **UTF-8** option in the Recording Options, you are forcing VuGen to record a Web page with the selected encoding, even when it uses different encoding. If, for example, a non-EUC encoded Web page is recorded as EUC-JP, the script will not replay properly.

**To enable the charset Encoding:**

1 Open the Recording Options (Ctrl+F7) and select the **Advanced** node.




2 Select **Support charset**. Choose the desired character encoding—UTF-8 or EUC-JP (only enabled for the Kanji operating system).

3 Click **OK**.

For more information about these settings, see "Setting Advanced Recording Options" on page 564.

### Manually Enabling Encoding

You can manually add full support for recording and replaying of HTML pages encoded in EUC-JP using the **web_sjis_to_euc_param** function. This also allows VuGen to display Japanese EUC-encoded characters correctly in Vuser scripts.

When you use **web_sjis_to_euc_param**, VuGen shows the value of the parameter in the Execution Log using EUC-JP encoding. For example, when you replay the **web_find** function, VuGen displays the encoded values. These include string values that were converted into EUC by the **web_sjis_to_euc_param** function, or parameter substitution when enabled in the **Run-Time Setting** > **Log** > **Extended Log**.

### Browser Configuration

If, during recording, non-English characters in the script are displayed as escaped hexadecimal numbers (For example, the string "Ü&" becomes "%DC%26"), you can correct this by configuring your browser not to send URLs in UTF-8 encoding. In Internet Explorer, choose **Tools > Internet Options and** click the **Advanced** tab. Clear the **Always Send URLs as a UTF-8** option in the Browsing section.

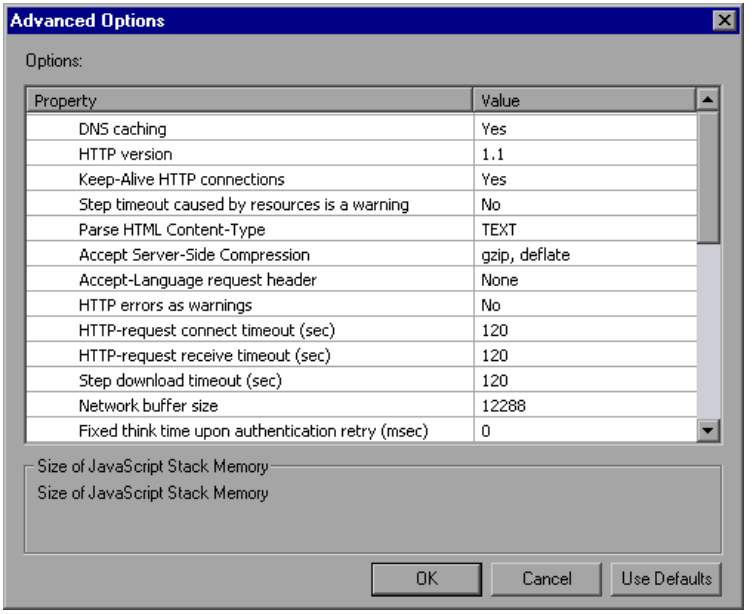
For more information about **web_sjis_to_euc_param**, refer to the *Online Function Reference*.

# Specifying a Language for the Accept-Language Header

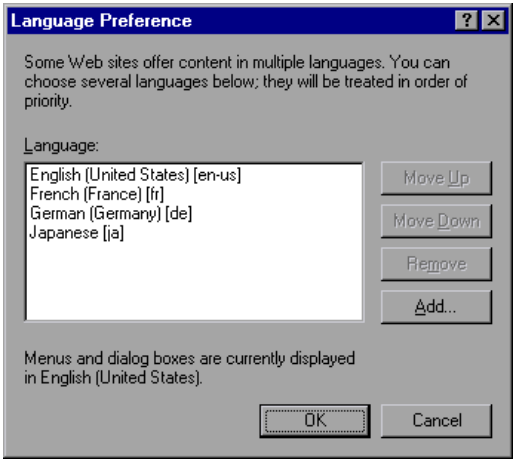
Before running a Web script, you can set the page's request header to match your current language. In the Internet Protocol Run-Time settings, you set the language of the *Accept-Language* request header. This header provides the server with a list of all of the accepted languages.

**To set the Accept-Language header:**

1. Open the Run-Time settings (F4) and select the **Internet Protocols:Preferences** node.
2. In the Advanced section, click the **Options** button to open the Advanced Options dialog box.



3 Locate the **Accept-Language request header** option. In the **Value** column, select the desired language from the list. This list is derived from the Internet Options Language settings in your browser.




For more information about these settings, see “Additional Options for Internet Preferences” on page 604.

# Protocol Limitations

### SMTP Protocol

If you work with SMTP protocol through MS Outlook or MS Outlook Express, the Japanese text recorded in a Vuser script is not displayed correctly. However, the script records and replays correctly.

### Script Name Length

When recording in COM, FTP, IMAP, SMTP, POP3, REAL or VB in VBA mode, the length of the script name is limited to 10 multi-byte characters (21 bytes).

## Quality Center Integration

To open a script saved in a Quality Center project from VuGen, or a scenario saved in a Quality Center project from Controller, add a new Test Set named "Default" (in English) to the Quality Center project.

# C

# Programming Scripts on UNIX Platforms

Vusers on UNIX platforms can create scripts through programming. To create a script through programming, you use a template.

This appendix describes:

- About Programming Vuser Scripts to Run on UNIX Platforms
- Generating Templates
- Programming Vuser Actions
- 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 Vuser template as a basis for a larger Vuser scrips. The template provides:

- correct program structure
- Vuser API calls
- source code and makefiles for creating a dynamic library

After creating a basic script from a template, you can enhance the script to provide run-time Vuser information and statistics. For more information, see Chapter 7, "Enhancing Vuser Scripts."

## Generating Templates

VuGen includes a utility that copies a template into your working directory. The utility is called *mkdbtest*, and is located in $M_LROOT/bin. You run the utility by typing:

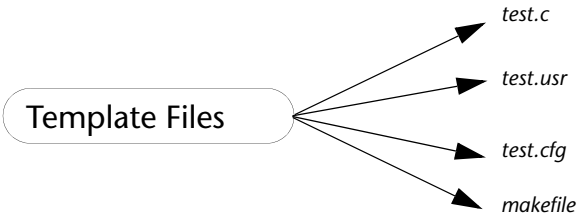
```
mkdbtest name
```

When you run *mkdbtest*, it creates a directory called *name*, which contains the template file, *name.c*. For example, if you type:

```
mkdbtest test1
```

*mkdbtest* creates a directory called *test1*, which contains the template script, *test1.c*.

When you run the *mkdbtest* utility, a directory is created containing four files *test.c*, *test.usr*, *test.cfg* and *Makefile*, where *test* is the test name you specified for mkdbtest.




## Programming Vuser Actions

The Vuser script files, *test*.c, *test*.usr, and *test*.cfg, can be customized for your Vuser.

You program the actual Vuser actions into the *test.c* file. This file has the required structure for a programmed Vuser script. The Vuser script contains three sections: vuser_init, Actions, and vuser_end.

Note that the template defines *extern C* for users of C++. This definition is required for all C++ users, to make sure that none of the exported functions are modified inadvertently.

```
#include “lrun.h”
#if defined(__cplusplus) || defined(cplusplus) extern “C”
{
#endif
int LR_FUNC vuser_init(LR_PARAM p)
{
                lr_message(“vuser_init done\n”);
                return 0;
}
 int Actions(LR_PARAM p)
{
                lr_message(“Actions done\n”);
                return 0;
}
 int vuser_end(LR_PARAM p)
{
                lr_message(“vuser_end done\n”);
                return 0;
}
#if defined(__cplusplus) || defined(cplusplus)}
#endif
```

You program Vuser actions directly into the empty script, before the **lr_message** function of each section.

The *vuser_init* section is executed first, during initialization. In this section, include the connection information and the logon procedure. The *vuser_init* section is only performed once each time you run the script.

The *Actions* section is executed after the initialization. In this section, include the actual operations performed by the Vuser. You can set up the Vuser to repeat the Actions section (in the *test*.cfg file).

The *vuser_end* section is executed last, after the all of the Vuser’s actions. In this section, include the clean-up and logoff procedures. The *vuser_end* section is only performed once each time you run the script.

**Note:** Mercury applications control 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 Chapter 12, "Configuring Run-Time Settings".) The *default.cfg* file contains the setting for the General, Think Time, and Log options. The *default.usp* file contains the setting for the Run Logic and Pacing.

### General Options

There is one General options for Unix Vuser scripts:

**ContinueOnError** instructs the Vuser to continue when an error occurs. To activate the option, specify 1. To disable the option, specify 0.

In the following example, the Vuser will continue on an error.

```
[General]
ContinueOnError=1
```

## Think Time Options

You can set the think time options to control how the Vuser uses think time during script execution. You set the parameters Options, Factor, LimitFlag, and Limit parameters according to the following chart.

| Option | Options | Factor | LimitFlag | Limit |
|---|---|---|---|---|
| Ignore think time | NOTHINK | N/A | N/A | N/A |
| Use recorded think time | RECORDED | 1.000 | N/A | N/A |
| Multiply the recorded think time by... | MULTIPLY | number | N/A | N/A |
| Use random percentage of recorded think time | RANDOM | range | lowest percentage | upper percentage |
| Limit the recorded think time to... | RECORDED / MULTIPLY | number (for MULTIPLY) | 1 | value in seconds |

To limit the think time used during execution, set the LimitFlag variable to 1 and specify the think time Limit, in seconds.

In the following example, the settings tell the Vuser to multiply the recorded think time by a random percentage, ranging from 50 to 150.

```
[ThinkTime]
Options=RANDOM
Factor=1
LimitFlag=0
Limit=0
ThinkTimeRandomLow=50
ThinkTimeRandomHigh=150
```

## Log Options

You can set the log options to create a brief or detailed log file for the script's execution.

```
[Log]
LogOptions=LogBrief
MsgClassData=0
MsgClassParameters=0
MsgClassFull=0
```

You set the parameters LogOptions, MsGClassData, MsgClassParameters, and MsgClassFull variables according to the following chart:

| Logging Type | LogOptions | MsgClassData | MsgClassParameters | MsgClassFull |
|---|---|---|---|---|
| Disable Logging | LogDisabled | N/A | N/A | N/A |
| Standard Log | LogBrief | N/A | N/A | N/A |
| Parameter Substitution (only) | LogExtended | 0 | 1 | 0 |
| Data Returned by Server (only) | LogExtended | 1 | 0 | 0 |
| Advanced Trace (only) | LogExtended | 0 | 0 | 1 |
| All | LogExtended | 1 | 1 | 1 |

In the following example, the settings tell the Vuser to log all data returned by the server and the parameters used for substitution.

```
[Log]
LogOptions=LogExtended
MsgClassData=1
MsgClassParameters=1
MsgClassFull=0
```

## Iterations and Run Logic

You can set the Iteration options to perform multiple iterations and control the pacing between the iterations. You can also manually set the order of the actions and their weight. To modify the run logic and iteration properties of a script, you must edit the *default.usp* file.

To instruct the Vuser to perform multiple iterations of the Actions section, set RunLogicNumOfIterations to the appropriate value.

To control the pacing between the iterations, set the RunLogicPaceType variable and its related values, according to the following chart:

| Pacing | RunLogicPaceType | Related Variables |
|---|---|---|
| As soon as possible | Asap | N/A |
| Wait between Iterations for a set time | Const | RunLogicPaceConstTime |
| Wait between iterations a random time | Random | RunLogicRandomPaceMin, RunLogicRandomPaceMax |

| Pacing | RunLogicPaceType | Related Variables |
|---|---|---|
| Wait after each iteration a set time | ConstAfter | RunLogicPaceConstAfterTime |
| Wait after each iteration a random time | After | RunLogicAfterPaceMin, RunLogicAfterPaceMax |

In the following example, the settings tell the Vuser to perform four iterations, while waiting a random number of seconds between iterations. The range of the random number is from 60 to 90 seconds.

```
[RunLogicRunRoot]
MercIniTreeFather=""
MercIniTreeSectionName="RunLogicRunRoot"
RunLogicRunMode="Random"
RunLogicActionOrder="Action,Action2,Action3"
RunLogicPaceType="Random"
RunLogicRandomPaceMax="90.000"
RunLogicPaceConstTime="40.000"
RunLogicObjectKind="Group"
RunLogicAfterPaceMin="50.000"
Name="Run"
RunLogicNumOfIterations="4"
RunLogicActionType="VuserRun"
RunLogicAfterPaceMax="70.000"
RunLogicRandomPaceMin="60.000"
MercIniTreeSons="Action,Action2,Action3"
RunLogicPaceConstAfterTime="30.000"
```

## Defining Transactions and Rendezvous Points

When programming a Vuser script without VuGen, you must manually configure the Vuser file in order to enable transactions and rendezvous. The configuration settings are listed in the *test*.usr file.

```
[General]
Type=any
DefaultCfg=Test.cfg
BinVuser=libtest.libsuffix
RunType=Binary

[Actions]
vuser_init=
Actions=
vuser_end=

[Transactions]
transaction1=

[Rendezvous]
Meeting=
```

Each transaction and rendezvous must be defined in the *usr* file. Add the transaction name to the Transactions section (followed by an "="). Add each rendezvous name to the Rendezvous section (followed by an "="). If the sections are not present, add them to the *usr* file as shown above.

## Compiling Scripts

After you modify the template, you compile it with the appropriate *Makefile* in the script's directory. Note that for C++ compiling, you must use the native compiler (not gnu). The compiler creates a dynamic library called:

- libtest.so (solaris)
- libtest.a (AIX)
- libtest.sl (HP)

You can modify the *Makefile* and assign additional compiler flags and libraries by modifying the appropriate sections.

If you are working with a general template, you must include your application's libraries and header files. For example, if your application uses a library called *testlib*, include it in the LIBS section.

```
LIBS        = \
      -testlib \
      -lLrun50 \
      -lm
```

After you modify the *makefile*, type Make from the command line in the working directory to create the dynamic library files for the Vuser script.

After you create a script, you check it's functionality from the command line.

To run a Vuser script from the UNIX command line, type:

```
mdrv -usr 'pwd' test.usr
```

where *pwd* is the full path to the directory containing the Vuser script and *test*.usr is the name of the Vuser file. Check that your script communicates with the server and performs all the required tasks.

After you verify that your script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, Tuning Module session, or Business Process Monitor profile. For more information, refer to the *LoadRunner Controller User's Guide*, *Tuning Console*, *Performance Center*, or *Business Availability Center* documentation.

# D

# Using Keyboard Shortcuts

The following list describes the keyboard shortcuts available in the Virtual User Generator.

| | |
|---|---|
| ALT+F8 | Compares the Current Snapshots (Web Vusers only) |
| ALT+INS | Create New Step |
| CTRL+A | Select All |
| CTRL+C | Copy |
| CTRL+F | Find |
| CTRL+G | Go To Line |
| CTRL+H | Replace |
| CTRL+N | New |
| CTRL+O | Open |
| CTRL+P | Print |
| CTRL+S | Save |
| CTRL+V | Paste |
| CTRL+X | Cut |
| CTRL+Y | Redo |
| CTRL+Z | Undo |
| CTRL+F7 | Recording Options |
| CTRL+F8 | Scan for Correlations |

| | |
|---|---|
| CTRL+SHIFT+SPACE | Show Function Syntax (Intellisense) |
| CTRL+SPACE | Complete Wizard (completes the function name) |
| F1 | Help |
| F3 | FIND Next Downward |
| SHIFT+F3 | Find Next Upward |
| F4 | Run-Time Settings |
| F5 | Run Vuser |
| F6 | Move Between Panes |
| F7 | Show EBCDIC Translation Dialog (for WinSocket data) |
| F9 | Toggle Breakpoint |
| F10 | Run Vuser Step by Step |

# Index


## A

ABC icon 117
accept Server-Side compression 605
Accept-Language request header 605, 1188
Action
    method 431
    section 54
Action steps
    function list-Web 549
    modifying - Web 639
actions
    importing 71
    recording multiple 65
    reordering 72
Actions class 429
ActiveX, enabling support 608
Add new column dialog box 135
Add Rule 683
Add/Remove dialog box, object identification 842
Additional Attributes run-time setting 191
ADO .NET Vuser 521
Advanced GUI dialog box 863
Advanced recording options 564
agent, for Citrix 337
allocating Vuser values
    data files 141
    data tables 144
ALNUM flag 629
animated run
    defined 205
    enabling 206
ANSI 988
ANSI C support, in custom scripts 421
Application Deployment Solution, Citrix Vuser type 303–336
application server, Oracle NCA 870
argument view 775
AssignToParam property (Web) 633
assistive properties, configuring 841
authentication during recording 72
authentication retry think time 606
auto-detect protocol 88
automatic proxy configuration script 595
automatic transactions
    Database Vuser scripts 352
    general 197
    Web and Wireless protocols 604
automation compliant 418
autorecovery 15

## B

Basic event recording configuration level 847
bearers
    support (WAP) 1091
behavior, DHTML 854
BIN flag 629
binary coded data 636
binary view of data (WinSock) 391
block size, allocating Vuser values 141
bookmarks in a Vuser script 214
bookmarks in data (WinSock) 394
boundaries, defining for correlation 692
braces, using in parameterization 129
Breakpoint Manager 212
breakpoints 210
Brief log run-time setting 187
browser
    cache (Web, Wireless) 600
    launching (Web/WinSock) 793
    manual launching (Web) 574
    recording options (Web) 574
    specifying location (Web) 574

Browser Emulation settings, Web 598
bubbling 855
buffer navigator (WinSock) 392
buffer size on network (Internet) 606

## C

C functions
  additional keywords 1138
  for debugging 1138
  using in Vuser scripts 30
C language support
  conventions 421
  interpreter 31
C Vusers 419
cache
  check for newer versions 600
  clear each iteration 601
  loading and dumping 555
CARRAY buffers 1060
character encoding 1185
character set, RTE 988
Check In command 235
Check Properties dialog box 660
checking-in scripts 235
checks (Web)
  defining properties for 633
  functions 550
  image checks 630
  modifying in scripts 660
  overview 619
  text 622
  types of 621
cHTML 1078
Citrix agent 337
Citrix server, disconnecting 307
Citrix Vuser scripts 303–336
  client version 306
  disconnecting from server 307
  display settings 317
  editing 321
  function list 329
  getting started 305
  recording options 316
  run-time settings 318
  synchronizing replay 322
  tips for record and replay 333
Classpath
  recording options 271
  run-time settings 299
client emulation
  Oracle NCA 883
  Web Services 768
client for Citrix 306
client request view 775
client-side certificates 711
Close All command 217
CLR Vusers 521
code generation options (EJB) 815
Collapse All command 706
COM
  data types 452
  overview and interfaces 450
COM Vuser scripts
  class context 451
  creating object instances 468
  debug functions 489
  developing 449
  error checking 467
  functions 479
  functions that create instances 480
  getting started 452
  IDispatch interface 472, 481
  instantiating objects 468
  interface pointers 466
  log files for debugging 453
  lrc_*type* functions 481
  parameterization functions 484
  recording options 456, 524
  retrieving an interface 469
  scanning for correlations 474
  script structure 466
  selecting COM objects to record 453
  type assignment functions 481
  type libraries 451
  understanding 466
  variant type conversion functions 482
  Visual Basic collection 488
command line arguments
  parsing functions 33
  reading in C Vuser scripts 108
  reading in Java Vuser scripts 440

UNIX Vuser scripts 218
command prompt 217
Comment button 100
comments
adding to correlation steps 668
inserting into Vuser scripts 100
screen header comments (RTE) 991
comparing
WSDL files 769
XML files 773
comparing Vusers 170
comparison method 681, 722
compiling Vuser scripts (UNIX) 1200
compression for HTML (gzip) 611
compression headers, requesting 607
concurrent group functions 551
configuration levels
customizing 849–859
standard 847–848
Connect dialog box (RTE) 984
connection attempts, modifying (RTE) 1001
Connection to Quality Center dialog box 226
content check
limit errors 606
settings (Web) 614
content type filtering (Web) 568
Content type filters dialog box 569
Continue on error 105, 193
Control steps
functions (Web) 554
modifying (Web) 656
Controller
scenario 222
converting
custom request to C 653
Web functions to Java 544
copy and paste
advanced for WinSock Vusers 397
RTE Vusers 986
Corba Recording Options dialog box 282
Corba-Java Vuser scripts 491
CORBA recording options 282
correlation options 277
debug options 279
Recorder options 272
recording 492
serialization options 276
Correlated Query tab
COM 475
Database 372
correlating
advanced properties 671
after recording (Web, Wireless) 675
COM Vusers 474
debugging tips for Siebel 1152
for known contexts (Web) 664
functions (C) 167
functions (Java) 169
HTML statements (Web) 661
Java statements 285
maximum parameter size 665
modifying existing parameters 172
overview 165
recording options-Java 277
rules for Web Vusers 665
scanning Database Vuser script 372
scripting language options 80
Siebel-Web 664, 955
SWECount 963
Tuxedo 1063
with snapshots (Web) 675
Correlation options, for Script recording options 80
Correlation Results tab 683
Correlation tab 673
Create Rule 678
CtLib 347
logging server messages 188
options 354
result set errors 369
custom event-recording configuration 849–859
adding listening events 853
adding objects to the list 852
deleting objects from the list 852
procedure 849
specifying listening criteria 854
custom headers, for Web and Wireless 566
Custom Request dialog box (Web) 654

Custom Request step
defined 555
for XML 698
modifying (Web) 653
custom requests 587
Custom Vuser types
C Vusers 419
Java Vusers 423
JavaScript Vusers 426
VB Vusers 424
VBScript Vusers 425
custom web event configuration files
loading 858
saving 857
Custom Web Event Recording Configuration dialog box 849

## D

data assignment methods, in parameterization 144, 146
data buffers
Tuxedo Vuser scripts 1056
WinSock Vuser scripts 402
data file parameters
adding rows and columns 135
creating a data source 134
editing 136
importing data from database 136
importing data source using data wizard 135
selecting data source 134
data files
used for parameterization 120
Windows Sockets Vuser scripts 404
data grids, enabling 217, 530
data table parameters
adding rows and columns 135
creating a data source 134
editing 136
importing data from database 136
importing data source using data wizard 135
selecting data source 134
Data Wizard 137
MS Query 137
SQL statement 138
Database Query Wizard dialog box 137
Database recording options 352
Database Vuser scripts
correlating 371
developing 347
getting started 351
handling errors 368
return codes 367
row information 365
tips 1147
using lrd functions 356
viewing grids 364
date/time, parameter values 151
DB2-CLI 347
DBCS 988
DbLib 347
DCOM 450
DCOM tab 458
Debug Message dialog box 104
Debug recording settings (Java) 279
debugging
database applications 1139
during replay 210
enabling debugging features 216
enabling for Web Vusers 216
obtaining information (WAP) 610
Oracle applications 1141
setting debug level 187
Web Vuser scripts 703
decrypting text 109
deep correlation (Java) 287
defining parameter properties
data files 140
general 122
tables 142
defining properties, text checks 622
deleting
objects from list 852
deleting steps
from Web scripts 638
delimiter of columns
in data files 140
in data tables 142

detector, EJB 806
device name (RTE) 1002

diagnostics, enabling in VuGen 198
DIG flag 629
disable logging log option 185
disabling functions (SAPGUI) 925
disconnecting from Quality Center 228
Display tab, General options 216
distinguished names 519
DLLs, calling from a Vuser script 1177
DN (LDAP) 519
DNS caching
    Web 604
DNS Vusers
    functions 378
    overview 377
DOM memory allocation 607
download filter 608
downloading
    from Performance Center to VuGen 249
DSL 202
dual protocol
    SAPGUI/SAP-Web 895
    Web/WinSock 789
duplicate key violations
    Oracle, MSSQL 1149
    Siebel 1152
dynamic ports 412, 535

## E

EBCDIC translation 406
editor, setting font for 15
EJB
    code generation options 815
    instance 818
    method 820
    Vuser scripts 805
EJB Detector 818
    command-line 807
    log files 810
    setup 806
encoding passwords 110
encoding, EUC 587
encrypting text 109
end method 430
End Transaction dialog box 98
engine, recording 565
environment settings
    Java 441
    Tuxedo Vusers 1057
Environment tab 15
error handling 105, 193
    COM Vuser scripts 467
    modifying globally 368
    modifying locally (severity level) 369
error matches, limiting 606
Error Message dialog box 104
errors, generate snapshot on 193
escape sequence 408
EUC encoded pages 587
EUC encoding 589
EUC-JP encoding
    recording option 565
    setting 1185
event-recording configuration 846–859
    customizing levels 849
    resetting 858
    standard levels 847
Execution report (Web only) 722
Expand All command 706
Expect property, Web checks 634
exporting to a zip file 67
extended log option 186
extended result set 355
external functions 1177

## F

fetching data 365
field demarcation characters 997
FIELDTBLS environment setting 1057
files, adding to script 111
filtering
    content type (Web, Wireless) 568
    downloaded resources 608
    report information (Web) 707
Filters dialog box (Web reports) 707
flags, text search 629
FLDTBLDIR environment setting 1057

font in editor 15
format
  for parameterization 161
  of data in display buffer 408
Forms Listener 886
Frame property, for object checks (Web) 633
FTP protocol
  function list 512
  recording 511
full run-time trace 187
functions
  automatic word completion 37
  ctrx (Citrix) 329
  DNS 378
  FTP 512
  GUI level (PeopleSoft, Oracle) 832
  imap 1026
  in Web Vuser scripts 548
  Java 432
  list of C functions 32
  lr (C functions) 29
  lrc (COM) 465
  lrd (Database) 356
  lreal (Real Player) 1070
  lrs (WinSock) 385
  lrt (Tuxedo) 1048
  mapi 1028
  mms 1071
  pop3 1030
  sapgui (SAP) 929
  smtp 1031
  syntax 38
  te (RTE) 975
  WAP 1087

## G

Gateway settings (WAP) 1106
General options
  all Vusers 130
  Citrix display 317
  Correlation tab 680
  dialog box 131
  Display tab (Web only) 216
  Environment tab 15
  Parameterization tab 129
  Replay tab 206
Generate snapshot on error 193
Generation Log tab 70
Get Text tool, Citrix Vuser scripts 341
global directory 130
Global Unique Identifier (GUID) 451
go to command 215
graphs
  enabling for Web 603
grids
  hiding 217, 530
  viewing 364
group name, parameter values 153
GUI Vuser scripts
  tools for 9
GUID 451
gzip 611

## H

handler 854
header files 38
headers
  custom 566
  risky 566
High event recording configuration level 848
history object, support for 608
host suffix, filtering by 608
HotSync 800
HTML
  maximum parameter length 693
HTML view (Web snapshots) 22
HTML-based mode 575
HTTP
  buffer size (Web) 606
HTTP recording mode, WAP 1106
hypergraphic link step, Web Vusers 555
hypertext link step
  defined 554
  modifying 641

## I

IC flag 629
ica files 328
IDE Integration, Web Services 765

IDispatch interface 472, 481
If-Modified-Since header
  Web 600
IIOP 502
image checks
  modifying (Web) 643
  Web Vuser scripts 630
Image Step Properties dialog box 644
IMAP protocol 1023
i-mode
  overview 1078
  toolkits 1079
importing
  actions 71
  data from a database 136
Informix 347
init method 430
Insert Comment dialog box 100
Instantiating COM objects 468
intellisense 37
internal data, parameterization 121, 150
Internet Messaging (IMAP) 1023
ISDN 202
iteration number, parameter values 154
iterations
  run-time settings 182
  updating parameters for each 162
IUnknown interface 451

## J

Jacada Vuser scripts 1035
  recording 1038
  replaying 1040
  understanding 1041
Java plug-in 498
Java virtual machine
  recording options 269
  run-time settings 298
Java Vusers (all)
  correlating statements 285
  editing Java methods 429
  environment settings 441
  inserting rendezvous points 436
  programming 427
  run-time settings 297–300
Java Vusers (Corba, RMI)
  Classpath run-time settings 299
  Java VM run-time settings 298
  recording options, correlation 277
  recording options, Java VM 269
  recording options, serialization 276
  recording tip 498
Java Vusers (custom)
  creating template 429
  using Java code 423
JavaScript Vusers 426
JNDI properties
  advanced, context factory 813
  locating EJB home 816
  specifying 812
Jscript 78

## K

KDC (Key Distribution Server) 607
keep-alive connections, Web 605
Kerberos
  server address 607
Kerebros
  authentication 607
keyboard mapping (RTE) 977
keyboard shortcut
  recording options 81
  run-time settings 174
  shortcuts list 1203
keywords, adding additional 1138

## L

language encoding 1185
language for script generation 77
language headers 1188
LDAP protocol
  function list 516
  recording 515
  via WinSock 379
libc functions, calling 421
libraries, for scripting 198
license information 8
Link Step Properties dialog box 641
load balancing, Oracle NCA 889

load generator name, parameter values 155
loading DLLs
    globally 1179
    locally 1177
    overview 1177
log
    setting detail level - PC 186
    setting detail level - UNIX 1197
log cache size 186
Log Message dialog box 103
Log run-time settings 184
lrbin.bat utility 1120
lrc functions 465
lrd (Database) functions 356
lreal functions 1070
lrs functions 385
lrt functions 1048

## M

Mailing Services protocols
    IMAP 1026
    MAPI 1028
    POP3 1030
    recording 1024
    SMTP 1031
mandatory properties, configuring 841
MAPI
    working with functions 1028
mapping keyboard 977
MatchCase property 633
maximum length of HTML parameters 693
Media Player 1071
Medium event recording configuration level 847
memory allocation for DOM 607
messages
    function list 35
    sending to output 101
META refresh 607
methods, Java 429
Microsoft .NET Vuser scripts
    developing 521
    getting started 523
Miscellaneous run-time settings 192
mkdbtmpl script (UNIX) 1192
MMS functions (MS Media Player) 1071
modifying Web scripts
    image steps 643
    rendezvous points 657
    submit data steps 649
    submit form steps 645
    think time 658
    transactions 656
    URL steps 639
MS
    Exchange protocol (MAPI) 1028
    SQL Server, recording on 347
MS Query 137
MTS components 461
multi action 56
multi protocol 56
multilingual support 1181–1190
    parameter files 1183
multithreading 196

## N

navigating through WinSock data 392
NCA Vusers, see Oracle NCA
NET references 528
network settings 202
Network Speed, run-time setting 202
New button 981
New Virtual User dialog box
    RTE 981
Nokia toolkits 1087
non-printable characters 409
non-resources 570
non-standard HTTP applications 720
NTLM authentication 72

## O

ODBC recording 347
offset of data in buffer (WinSock) 406
OnFailure property, Web checks 634
online browser 215, 718
Open command 709
optional windows 921
optional windows (SAPGUI) 925

options
CtLib 354
lrd log 354
recording (RTE) 989
Oracle
recording 2-tier database 347
version 8.0 359
Oracle application debugging 1141
Oracle Configurator 886
Oracle NCA Vuser scripts
check connection mode 887
correlating 889
creating 867
recording guidelines 870
run-time settings 883
secure applications 886
servlet testing 886
using Vuser functions 877
Oracle Web Applications 11i Vuser scripts
about 830
advanced GUI-based options 863
functions 832
recording level 859
tips 838
OrbixWeb 500
OTA, Over-The-Air 1093
Output Message dialog box 104
Output window 437
hiding 208
Replay tab 208
RunTime Data tab 209
show/hide 217

## P

Pacing settings 182
page view (Web Services) 775
page view (Web snapshots) 22
Palm
protocol 789
recording applications 800
PAP, Push Access Protocol 1093
parameter formats
adding 161
deleting 162
restoring original 162
Parameter Properties dialog box 122
parameter types
data files 120, 149
data tables 149
date/time 151
group name 153
internal data 121, 149, 150
iteration number 154
load generator name 155
random number 156
tables 120
understanding 119
unique number 157
user-defined functions 121, 149, 160
Vuser ID 159
parameterization
assigning values from files and tables 144
brace style 119, 129
COM, .NET, VB 116
creating a new parameter 116
data files 120
defining properties 122
global directory 130
internal data type formats 161
Java 117
limitations 115
modifying existing parameters 172
naming a parameter 117
overview 114
parameter list 127
random sequence with seed 145
restoring original value 126
setting properties for data files 140
setting properties for tables 142
tables 120
Tuxedo scripts 1055
undoing (Web) 126
updating parameter values 162
updating with unique values 145
user-defined functions 121
using internal data 121, 150
using user-defined functions 160
UTF-8 encoded 1183
Parameterization Options 129
parameterization, replacing long string 79

parameters
creating in Script view 116
creating in Tree view 117
creating using Parameter List 128
deleting 128
modifying 127
Password Encoder dialog box 110
password, encoding 110
pausing a Vuser 208
PeopleSoft Enterprise Vuser scripts
about 830
advanced GUI-based options 863
functions 832
recording level 859
tips 838
PeopleSoft-Tuxedo Vusers 1045
running 1156
Performance Center
connecting to 242
managing Vuser scripts 241
persistent connections, Web 605
phone, recording over 1090
POP3 (Post Office) protocol 1030
Port Mapping settings 84
PPG, Push Proxy Gateway 1093
Pragma mode 883
Preferences run-time settings 602
Print dialog box (Web reports) 709
printing Results Summary reports 709
programming
in Visual Studio 1115
using templates 1116, 1192
Vuser actions 1193
properties
AssignToParam (Web) 633
Expect (Web) 634
Frame (Web) 633
MatchCase (Web) 633
OnFailure (Web) 634
Repeat (Web) 634
Report (Web) 634
text checks 633
properties of parameters
defining 122
defining for data files 140
defining for tables 142
protocols, list of supported 60
Proxy Authentication dialog box 562
proxy server
recording options (Web) 560
recording options (Web/WinSock) 794
run-time settings (Internet) 593
Push support 1092

## Q

QC 225
Quality Center
connecting to 226
disconnecting from 228
managing scripts with 225
managing versions 232
managing Vuser scripts 225
opening a Vuser script 229
saving scripts to 231
version control for 232

## R

Radius
run-time settings (WAP) 1111
support 1092
random number, parameter values 156
random parameter assignment 145
read only WinSock buffers 390
RealPlayer 1067
Record button 63
recording
status, options 855
recording at the cursor 910
recording engine 565
Recording Log tab 69
recording mode
i-mode, VoiceXML 1099
WAP 1098
recording options
Advanced (Web, Wireless) 564
Browser (Web) 574
Corba Options 282
Database 352
Debug (Java) 279

Internet protocols 559
Java language 267–282
keyboard shortcut 81
Port Mapping 84
Recorder (Java) 272
Recording (Web) 588
Recording Level, GUI-based 859
Recording Proxy (Web, Wireless) 560
Recording Proxy (Web/WinSock) 794
RTE 989
RTE Configuration 988
Script (FTP, COM, Mail) 78
WAP Toolkit 1101
Web 573
WinSock 382
Wireless 1097
Wireless Vusers 1097
recording Vuser scripts
Corba-Java 492
Database 351
DNS 377
FTP 511
LDAP 515
Mailing services 1023
Oracle NCA 869
overview 53
proxy setting 560
Rmi-Java 501
SAPGUI 895
SAP-Web 945
Tuxedo 1045
Web/WinSock 789
Window Sockets 379
Wireless 1083
recovery of lost scripts 15
regenerating Vusers
all protocols 74
regression testing, WSDL 769
rendezvous
Rendezvous dialog box 99
rendezvous points
Java Vusers 436
modifying in Web scripts 657
rendezvous points, inserting 99
Repeat property, Web Vusers 634
Replace More Occurrences command 125
Replay tab, General Options dialog box 206
Report property, Web checks 634
Report toolbar 705
report tree, Results Summary (Web) 705
resources, excluding 570
restoring original value of parameter 126
Results Summary report 703
debugging Web scripts 703
filtering information 707
opening 708
printing 709
searching 708
tree branches 705
understanding 705
Web Services Vusers 785
return codes
database 367
RMI-Java Vuser scripts
correlation options 277
debug options 279
Recorder options 272
recording 503
recording over IIOP 502
serialization options 276
row count, obtaining 1149
row information, Database Vusers 365
RTE Vuser scripts
getting started 973
introducing 972
mapping PC keyboard 977
overview 971
reading text from screen 1017
recording 979
run-time settings 1000
steps in creating 973
synchronizing 1005
using te functions 975
rules
adding from Correlation tab 683
advanced correlation 668
creating from correlation results 678
defining for correlation 669
testing in correlation 672
Run command 207
Run Logic run-time settings 175
run_db_vuser shell script 218

running Vuser scripts
animated mode 205
step by step 210
using VuGen 203
run-time settings
Additional Attributes 191
all protocols 173
Browser Emulation node 598
Client Emulation (Oracle NCA) 883
Client Emulation (Web Services) 768
configuring manually 1195
ContentCheck node (Web) 613
debug information (WAP) 610
dialog box 174
Gateway node (WAP) 1106
Internet protocols (Web etc.) 591
Java 297–300
keyboard shortcut 174
Log node 184
Miscellaneous 192
NET references 528
network 201
Oracle NCA 883
Pacing node 182
Preferences - Advanced 603
Preferences (Internet prtcls) 602
Proxy (Internet prtcls) 593
Radius (WAP) 1111
RTE 1000
Run Logic 175
shortcuts 174
Speed Simulation 202
Think Time 189
VBA (Visual Basic Apps) 198
WAP 1105
run-time viewer
display options 207
enabling in VuGen 215
tips for using in VuGen 718

## S

S_SSA_ID table 1155
safearray log (COM) 464
SAPGUI Vuser scripts
auto logon recording options 914
code generation recording options 913
function list 929
general recording options 912
inserting steps 915
recording 895
recording at the cursor 910
run-time settings 925
setting recording options 912
snapshots 915
using sapgui functions 929
SAPGUI/SAP-Web dual protocol 895
SAP-Web Vuser scripts
recording 945
recording options 948
run-time settings 951
Scan for Correlations command
Database Vusers 372
scenario
create from VuGen 222
integrating Vuser scripts into 221
Script Generator, *See* VuGen
Search and Replace dialog box 125
searching for text on screen (RTE) 1018
sections of a Vuser script 54
secure WAP 1098
SED utility 544
Select or Create Parameter dialog box 116
Select Results Directory dialog box 206
sequential parameter assignment 144
serialization (Java correlation) 290
Serialization options 277
server response view 775
Service Step Properties dialog box 659
Service steps
modifying in tree view (Web) 659
session step
integrating Vuser scripts into 221
settings, *See* run-time settings
Shift Japan Industry Standard (SIJS) 587
show function 37
show function syntax 38
Siebel
base 36 key values 1155
scripting tips for 2-tier 1152
Siebel correlation library 955

Siebel-Web
correlating 664, 955
recording 954
troubleshooting 964
SMTP protocol 1031
snapshots
Citrix Vusers 321
generate on error 193
in Web Services scripts 775
SAPGUI Vusers 915
save replay snapshot locally 604
saving in Citrix recording 312
Web page 20
Winsock buffer 390
XML Vusers 696
Solaris
ASCII translations 383
Speed Simulation settings 202
split actions 78
SQL statement 138
standard event-recording configuration 847–848
standard log option 186
Start Recording dialog box 63
Start Transaction dialog box 97
Step button 210
stopping a Vuser 208
streaming data protocols
mms functions 1071
RealPlayer functions 1070
recording 1068
strings, replacing long with parameter 79
Submit Data step
defined 555
dialog box (Web) 650
modifying-Web 649
Submit Form step
defined 555
dialog box 646
modifying 645
suffix values in Siebel 1152
SWECount, correlating 963
synchronization functions
generating for Citrix text 313
synchronizing Vuser scripts
block-mode (IBM) terminals 1007
character-mode (VT) terminals 1010
overview 107
overview (RTE) 1005
waiting for terminal to be silent 1014
waiting for text to appear (RTE) 1012
waiting for the cursor to appear 1010
syntax, show for function 38
system variables
RTE 1012
Tuxedo 1057

## T

table icon 119
tables
used for parameterization 120
Tasks pane 42
te (RTE) functions 975
TE system variables 1012
template
creating new 61
Java Vuser 429
programming in "C" 1116, 1183, 1192
Terminal Emulation 979
Terminal Services
Citrix Vusers 333
Terminal Setup dialog box 983
test results
Web Services Vusers 785
Web Vusers 705
TestDirector, see Quality Center 225
text
comparing in snapshots (Web) 722
reading text from screen (RTE) 1018
searching for text on screen (RTE) 1018
text checks
defined 621
defining additional properties 633
flags 629
text synchronization, Citrix 313
text view (WinSock) 390
think time
defined 189
dialog box (Web treeview) 658
function (C) 35

function (Java) 434
inserting 107
modifying in Web scripts 658
recommended ratio for Siebel 967
run-time settings 189
threshold, Database 353
threshold, WinSock 385
Think Time dialog box 107
thread, main (Java programming) 445
thread-safe code 444
threshold for setTimeout, setInterval 607
thumbnails
annotating 26
in workflow wizard 25
renaming 26
viewing 24
Tile windows 217
timeouts
Citrix connection 320
HTTP request 606
WAP connection 606
timestamp (Database) 354
tips
Database related 1147
Siebel specific 1152
Token Substitution Testpad dialog box 672
token, parameterizing 666
traffic forwarding 87
Transaction Editor 25, 46
transactions
automatic, for LRD functions 352
automatic, for Web Vuser scripts 197
breakdown limitation for Oracle DB 55
editor 46
functions for 32
in output log 208
inserting 96
modifying in Web scripts 656
nested 50, 98
Web Vusers 197
wizard workflow 46
Translation table settings 383
translation, ASCII on UNIX 383
trapping 796
treeview
all Vusers 19
inserting steps 19
troubleshooting
2-tier database 1147
Oracle applications 1141
Siebel Vusers 1152
VuGen 1137
Web Vuser scripts 711
TUXDIR environment setting 1057
Tuxedo Vuser scripts 1045
data buffers 1056
environment settings 1057
log file 1055
running 1055
system variables 1057
Tuxedo 6, 7 1045
understanding 1053
using lrt (Tuxedo) functions 1048
versions 1058
viewing data files 1055
typing style (RTE Vusers) 994


## U


undo buffer, emptying (WinSock) 397
Undo Parameter command 126
unique column name 1152
unique number, parameter values 157
unique value parameter assignment 145
UNIX
command line 218
update methods, in parameterization 146, 162
uploading
required VuGen version 244
scripts to Performance Center 244
URL Step Properties dialog box
Web 640
URL steps
defined (Web Vusers) 554
modifying 639
URL-based mode 575
Use Existing Parameter command 124
user-agent browser emulation 599
user-defined function parameters 121

user-defined functions, parameterization 160
UTF-8 conversion
  recording option 565
  setting 1185

## V

validating
  wsdl files 755
validation
  wizard step 744
validation reports
  ws-i 762
validation, WSDL documents 761
validation, WS-I compliance 762
VB Vusers 424
VBA references 198
VBA run-time setting 198
VBScript Vusers 425
verification checks
  RTE 1017
  sapgui 921
  Web 603
verify_generator 218
version control 232
  checking tests in to 235
version history 237
Virtual User Generator, *See* VuGen
virtual users, defined 4
Visigenic 500
Visual Basic
  scripting option 78
  Vuser scripts 1115
Visual C, using Visual Studio 1115
Visual Log options (Web) 207
Visual Studio 1115
VM (virtual machine) 269
VoiceXML
  overview 1080
VuGen
  environment options 15
  introducing 13
  recording Vuser scripts 53
  running Vuser scripts 28
  starting 14
  toolbar 66
Vuser functions
  automatic word completion 37
  ctrx (Citrix) 329
  DNS 378
  external, user defined 1177
  FTP 512
  general (C) 29
  imap 1026
  Java 432
  list of C functions 32
  lr (C functions) 29
  lrc (COM) 468
  lrd (Database) 356
  lreal 1070
  lrs (WinSock) 385
  lrt (Tuxedo) 1048
  mapi 1028
  mms (MS Media Player) 1071
  Oracle NCA 877
  pop3 1030
  sapgui (SAP) 929
  smtp 1031
  syntax 38
  te (RTE) 975
  Web Services 783
  *See Also* Online Function Reference
Vuser Generator, *See* VuGen
Vuser ID, parameter values 159
Vuser information, obtaining 101
Vuser information, obtaining (Java) 437
Vuser scripts
  adding functions 93
  C support 421
  comments, inserting 100
  creating on UNIX 1191–1201
  custom 417
  debugging features 210
  enhancing 93
  importing from zip 62
  Java language recording 255
  opening 62
  parameterizing 113
  Performance Center
    upload/download 241
  programming 417, 1191–1201

Quality Center integration 225
regenerating 74
rendezvous points 99
running 203
running from command prompt 217
run-time settings 173
run-time settings-Java 297–300
scenario integration 221
sections 54
selecting generation language 77
session step integration 221
streaming data 1067
transactions 96
types *See* Vuser types
UNIX, compiling on 1200
UNIX, running on 218
uploading to Performance Center 244
version history 237
viewing 17
working from zip 62
Vuser types
COM 468
Corba-Java 491
EJB testing 805
Jacada 1035
Java (programming) 427
list of 7
list of (popup descriptions) 60
Media Player 1067
Real Player 1067
vuser_init, vuser_end sections 54
Vusers
introducing 4


## W


waiting, for terminal to stabilize(RTE) 1014
WAP Vuser scripts 1089–1095
debug information 610
introducing 1089
run-time settings 1105
specifying what to record 1098
Toolkit 1101
understanding 1089
Wdiff 170
Web correlation 661
Web Event Recording Configuration dialog box 848
Web functions, using 548
Web Gui-level scripts 829
Web performance graphs
generating for Web Vusers 603
Web Services
testing challenges 729
Web Services Vuser scripts
developing 735
functions 783
IDE integration 765
importing WSDL 743
parameterizing 781
recording 740
reporting tool 785
running 767
run-time settings 768
snapshots 775
testing 723
web_service_call 780
XML tree query 778
Web to Java converter 544
Web trapping 796
Web Trapping tab 797
Web Vuser scripts
adding steps 637
advanced tips 711
checks 619
content filtering 568
correlating 664
custom headers 566
custom request steps 653
debugging features, enabling 216
debugging tools 215
deleting steps 638
functions 545
image checks 630
Internet recording options 559
introducing 537
modifying 635
proxy settings 560
recording options 573
Results Summary report 703
run-time settings 201, 591
run-time viewer 215

sections 542
setting Visual Log options 215
specifying a browser for recording 574
understanding 539
verifying text and images 619
Visual Log options 207
Web/WinSock Vuser scripts 789
getting started 791
proxy settings 794
recording 798
Web trapping options 796
web_set_user function generation 72
Web-event-recording configuration 846–859
customizing 849–859
standard 847–848
window names, Citrix 312
Windows Sockets Vuser scripts
data buffers 402
data files 404
excluding sockets 384
getting started 380
recording 379
script and tree view 380
using lrs functions 385
viewing data files 402
working with data 389
WinInet engine (Internet protocols) 603
WinSock recording options 382
VoiceXML Vusers
*See* Wireless Vuser scripts
WAP Vuser scripts
*See Also* Wireless Vuser scripts
Wireless Vuser scripts
custom headers 566
getting started 1084
Internet recording options 559
introducing 1075
proxy settings 560
recording 1083
recording options 1097
understanding 1075
WAP toolkit 1101
wizard, workflow 41
word completion 37
workflow wizard 41
working copy, WSDL 744
WSDL documents
comparing 761, 769, 770
importing 743
management 754
overview 726
properties, setting 775
regression testing 769
tree view 775
validating 761
WS-I validation 762
WSP
recording options 1098
running mode 1106
session-recording over a phone 1090
WS*xxx* Tuxedo variables 1057


## X


XML
attributes, working with 1129
custom requests 698
editing tree in Web Services 764
testing 695
XML API
programming with 1121
XP window style, Citrix 310
X-SYSTEM message (RTE) 1007


## Z


zip file
exporting to 68
using 70
working from 62
zlib headers 607